BISS

*Programmer's Manual*

# *B-Wise*

## BML - User Interface Description

**BISS GmbH**

BML-Version 4.04 (11.06.2012)
Juni 2012

# *B-Wise* BML - XML User Interface Description

Die folgende Beschreibung stellt einen Auszug aus der *B-Wise*-Dokumentation dar.

***B-Wise* BML - XML User Interface Description, Version 4.04 (Stand: 11.06.2012)**

Die in dieser Unterlage enthaltenen Angaben und Daten können ohne vorherige Ankündigung geändert werden. Die in den Beispielen verwendeten Namen und Daten sind frei erfunden. Ohne ausdrückliche schriftliche Genehmigung der BISS GmbH darf kein Teil dieser Unterlagen für irgendwelche Zwecke vervielfältigt oder übertragen werden, unabhängig davon, auf welche Art und Weise oder mit welchen Mitteln, elektronisch oder mechanisch, dies geschieht. Alle Angaben und Informationen dieses Handbuchs wurden nach bestem Wissen und Gewissen verfasst. Trotzdem übernimmt die BISS GmbH keinerlei Gewährleistung für deren Richtigkeit oder Vollständigkeit.

© Copyright 2001 - 2012 - BISS GmbH, alle Rechte vorbehalten



© Copyright BISS GmbH, 2001-2012 All Rights reserved

# Inhaltsverzeichnis

**Inhaltsverzeichnis ..... 3**

**Kapitel 1 Einführung ..... 7**
1.1 Was ist BML? ..... 7
1.2 Warum BML ? ..... 14
1.3 Entwicklungsmethodik ..... 17

**Kapitel 2 BML Tutorial ..... 18**
2.1 *B-Wise* Benutzerschnittstelle ..... 18
2.2 Dialogseite ..... 21
2.3 Anzeigeseite ..... 33

**deafult Kapitel 3 BML-Sprachkonstrukte ..... 41**
3.1 Allgemeine Hinweise ..... 41
3.2 <bw:cjscript> ..... 42
3.3 <bw:element> ..... 44
3.4 <bw:foreach> ..... 45
3.5 <bw:group> ..... 47
3.6 <bw:if> ..... 48
3.7 <bw:ig> ..... 50
3.8 <bw:include> ..... 51
3.9 <bw:lookup> ..... 54
3.10 <bw:page> ..... 57
3.11 <bw:script> ..... 58
3.12 <bw:switch> ..... 59
3.13 <bw:usecls> ..... 60
3.14 <bw:var> ..... 62

**Kapitel 4 BML-Ausdrücke ..... 65**
4.1 Konstanten ..... 65
4.2 Interne Variablen ..... 66
4.3 Externe Variablen ..... 66
4.4 Operatoren ..... 66
4.5 Prädikate ..... 68

© Copyright BISS GmbH, 2001-2012 All Rights reserved

4.6 Funktionen....69
4.7 Lookup-Tabellen....76

**Kapitel 5 BML-Inheritance....77**

**Kapitel 6 BML und Java....82**

**Kapitel 7 BML und JavaScript....85**

**Kapitel 8 BML-Regeln (Rules)....88**
8.1 Einführung....88
8.2 <bw:copy>....92

**Kapitel 9 BML Dialogelemente....95**
9.1 Übersicht....95
9.2 Metadaten....98
9.3 Dialogbeschreibung....99
9.4 Internationalisierung....111
9.5 Text-Formate....112
9.6 Datenbindung....114
9.7 Daten-Formatierung....115
9.8 Elemente....117
9.8.1 <bw:label>....118
9.8.2 <bw:htmllabel>....125
9.8.3 <bw:button>....135
9.8.4 <bw:datetime>....141
9.8.5 <bw:mcalendar>....148
9.8.6 <bw:entry>....154
9.8.7 <bw:editor>....161
9.8.8 <bw:texteditor>....168
9.8.9 <bw:syntaxeditor>....174
9.8.10 <bw:browser>....181
9.8.11 <bw:checkbox>....189
9.8.12 <bw:choice>....196
9.8.13 <bw:list>....203
9.8.14 <bw:datagrid>....212
9.8.15 <bw:c_list>, <bw:h_list>....221
9.8.16 <bw:tree>....234
9.8.17 <bw:portal>....242
9.8.17.1 <bw:portletcontainer>....246
9.8.17.2 <bw:portletpane>....248
9.8.18 <bw:portlet>....250


© Copyright BISS GmbH, 2001-2012 All Rights reserved

9.8.19 <bw:barmenu>, <bw:contextmenu>, <bw:menu> ..... 255
9.8.20 <bw:compound> ..... 263
9.8.21 <bw:compoundbox> ..... 271
9.8.22 <bw:toolbar> ..... 277
9.8.22.1 <bw:tbbutton> ..... 285
9.8.22.2 <bw:tbcbutton> ..... 287
9.8.22.3 <bw:tbmbutton> ..... 289
9.8.22.4 <bw:tblabel> ..... 291
9.8.22.5 <bw:tbchoice> ..... 293
9.8.22.6 <bw:tbentry> ..... 295
9.8.22.7 <bw:tbseparator> ..... 297
9.8.23 <bw:radiobutton> ..... 299
9.8.24 <bw:notebook> ..... 307
9.8.24.1 <bw:bookpage> ..... 314
9.8.24.2 <bw:booktab> ..... 316
9.8.25 <bw:progress> ..... 318
9.8.26 <bw:slider> ..... 324
9.8.27 <bw:picturebox> ..... 331
9.8.28 Deco-Elemente ..... 337
9.8.28.1 <bw:line> ..... 338
9.8.28.2 <bw:rect> ..... 340
9.8.28.3 <bw:groupbox> ..... 342
9.8.28.4 <bw:circle> ..... 344
9.8.28.5 <bw:img> ..... 346
9.8.28.6 <bw:text> ..... 348
9.9 Styles ..... 350
9.10 BML-Widget-Beispiel ..... 356
9.11 *B-Wise* Layouter ..... 357
9.12 PopUp-Dialoge (modal bzw. nicht-modal) ..... 361
9.13 Scripting (Client-seitiges) ..... 363
9.13.1 Hinweise zur Verwendung von Scripten ..... 365
9.13.2 <bw:cjscript> ..... 367
9.13.2.1 Objekte und Methoden ..... 370
9.13.2.2 Objekt und Eigenschaften ..... 378
9.13.2.3 Script-Beispiele ..... 380

**Anhang A BML-Diskussion ..... 381**
Diskussion der BML innerhalb von *B-Wise* ..... 381

**Anhang B BML-DTD ..... 385**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Anhang C BML - Formale Definition ..... 388**

**Anhang D Beispiel-Templatedatei ..... 391**

Beispiel-Templatedatei (BWisePortal.BML) ..... 391

**Anhang E *B-Wise* SmartClient-Einstellungen ..... 393**

BWSmartClientCore_Com.config ..... 393
BWSmartClientCore_Portal.config ..... 394
BWSmartClientApp_User.config ..... 396
BWSmartClientApp_Menu.config ..... 396
BWSmartClientApp_Login.config ..... 400
BWSmartClientApp_Help.config ..... 401
BWSmartClientApp_Warning.config ..... 402
BWSmartClientApp_Error.config ..... 403

**Anhang F Attribut-Evaluation ..... 405**

**Anhang G BML-Dialogelemente ..... 406**

**Anhang H *B-Wise* Namenskonventionen ..... 408**

**Anhang J Versionsinformationen ..... 409**

Bekannte Fehler ..... 418


© Copyright BISS GmbH, 2001-2012 All Rights reserved

BISS

Kapitel 1
# Einführung

## 1.1 Was ist BML?

Die *B-Wise* XML User Interface Description (im Folgenden kurz BML genannt) ist Teil der technischen Basis des Application-Frameworks *B-Wise* und dient der abstrakten, plattform-neutralen Definition der Benutzerschnittstelle (also von Dialogen, Anzeigeseiten und Dokumenten).

BML gliedert sich in die folgenden Bereiche:

- Definition von Dialogelementen („Controls“)
- Sprachkonstrukte (Sprachelemente und -ausdrücke)
- Definition von Textdokumenten
- Regeln (Rules bzw. manchmal auch „Makros“ genannt)
- Schnittstellen zu Java und JavaScript

***Dialogelemente***
BML bietet einen umfangreichen Satz an Dialogelementen wie z.B. Eingabefelder, Buttons, Listen, Editoren oder Menüs. Diese Beschreibungen sind Plattform-neutral und erlauben damit die Unterstützung unterschiedlicher Clients (z.B. Windows-PC oder Apple iPad) mit einer einzigen Quelle (Sourcecode). Selbstverständlich werden auch Plattform-spezifische Konstrukte unterstützt.

***Sprachkonstrukte***
Mit Hilfe der Sprachkonstrukte bietet BML vielfältige Möglichkeiten zur Strukturierung des Quellcode des User Interface sowie zahlreiche funktionale Elemente zur Implementierung auch komplexer, dynamischer Dialoge, Anzeigeseiten und Dokumente.

Die Sprachkonstrukte von BML umfassen u.a. Bedingungen und Schleifen. Desweiteren erlauben Variablen und Funktionen den Zugriff auf das *B-Wise* DOM (Document Object Model) und damit die Ausgabe bzw. Darstellung dynamischer Inhalte. Schließlich bietet BML Unterstützung zur Strukturierung wie z.B. Aggregation oder Ableitung.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### *Textdokumente*

BML bietet zahlreiche Elemente für die Unterstützung des Dokumenten-Drucks. Diese erlauben, zusammen mit der optionalen Definition von Texten in Java bzw. RTF eine flexible Darstellung auch komplexer Dokumente. BML für Textdokumente ist nicht Gegenstand dieses Manuals sondern in einem eigenen Manual beschrieben.

### *Regeln*

Transformationsregeln (Rules) bieten eine Erweiterbarkeit des Elementumfangs durch Definition von Makros bzw. Unterstützung unterschiedlicher Endgeräte. Sie erlauben zudem eine individuelle Erweiterung des Sprachumfangs und damit optimale Anpassung an spezifische Projekte.

Zusammen mit weiteren Komponenten des Basisframeworks *B-Wise* stellt BML einerseits ein mächtiges andererseits ein einfach zu handhabendes Werkzeug für die Entwicklung von Benutzerschnittstellen (Dialoge und Dokumente) dar.

Ein Beispiel soll das bisher Gesagte veranschaulichen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bw:portlet
  xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml"
  id="Mobile_Server_AdminSearchUser"
  title="Benutzer und zugeordnete Geräte suchen und anzeigen"
  target="cont">

  <bw:include src="Mobile_Server_Standards.pts" />
  <bw:include src="Mobile_Server_DialogMacros.inc" />

  <bw:compound id="CP_Main" tabindex="1">
    <bw:pos aox="0" aoy="0" acx="1000" acy="1000" eox="0" eoy="0" ecx="0" ecy="0" />

   <bw:label id="LA_Head1" align="MiddleLeft">
      <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="10" eoy="5" ecx="300" ecy="25" />
      <bw:style>
         <bw:color>white</bw:color>
         <bw:font>Tahoma,10,bold</bw:font>
         <bw:backgroundcolor>#<bw:var expr="$S_bg2" /></bw:backgroundcolor>
      </bw:style>
      <bw:content>Benutzer suchen</bw:content>
   </bw:label>

   <bw:label id="LA_Info" align="TopLeft">
      <bw:pos rel="yflow" asmode="height" aox="0" aoy="0" acx="1000" acy="0"
                        eox="10" eoy="5" ecx="-10" ecy="5" />
      <bw:style>
         <bw:font>Tahoma,10,regular</bw:font>
      </bw:style>
      <bw:content>Bitte geben Sie die gewünschten Suchparameter ein. Die Parameter
         werden mit einem "UND" verknüpft, d.h. jede Eingabe schränkt das Ergebnis
         weiter ein. Sie können Platzhalter verwenden; ein "*" (Sternchen) steht dabei
         für beliebige und beliebig viele Zeichen, ein "?" (Fragezeichen) für genau
         ein beliebiges Zeichen.</bw:content>
   </bw:label>

   <bw:compound id="CP_Dlg" tabindex="1">
      <bw:pos rel="yflow" asmode="height" aox="0" aoy="0" acx="1000" acy="0"
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
                        eox="0" eoy="0" ecx="0" ecy="5" />

      <bw:compound id="CP_DlgL" tabindex="1">
         <bw:pos asmode="height" aox="0" aoy="0" acx="500" acy="0"
                        eox="0" eoy="0" ecx="0" ecy="5" />

         <bw:var name="tab"   expr="1" />
         <bw:var name="alg"   expr="'MiddleRight'" />
         <bw:var name="mleft" expr="200" />
         <bw:var name="mtop"  expr="5" />
         <bw:var name="defw"  expr="320" />
         <bw:dlgrow id="SP_1" type="tf"   label="Benutzername" datasrc="" />
         <bw:dlgrow id="SP_2" type="tf"   label="Vollständiger Name" datasrc="" />
         <bw:dlgrow id="SP_3" type="tf"   label="Wohnort" datasrc="" />
      </bw:compound>

      <bw:compound id="CP_DlgR" tabindex="1">
         <bw:pos rel="xflow" asmode="height" aox="0" aoy="0" acx="1000" acy="0"
                        eox="0" eoy="0" ecx="0" ecy="5" />

         <bw:var name="tab"   expr="1" />
         <bw:var name="alg"   expr="'MiddleRight'" />
         <bw:var name="mleft" expr="200" />
         <bw:var name="mtop"  expr="5" />
         <bw:var name="defw"  expr="320" />
         <bw:dlgrow id="SP_4" type="ch"   label="Gerätetyp" datasrc="" />
         <bw:dlgrow id="SP_5" type="tftf" label="Aktiviert von" label2="Aktiviert bis"
               wid="100" datasrc="" datasrc2="" />

         <bw:button id="BU_Search" tabindex="20" def="true">
            <bw:pos rel="yflow" aox="0" aoy="0" acx="1000" acy="0"
                        eox="420" eoy="5" ecx="-10" ecy="31"/>
            <bw:content><bw:t>Benutzer ~suchen</bw:t></bw:content>
         </bw:button>
      </bw:compound>
   </bw:compound>

   <bw:browser id="BR_1" tabindex="1">
       <bw:pos rel="yflow" aox="0" aoy="0" acx="1000" acy="1000"
                        eox="10" eoy="10" ecx="-10" ecy="-40" />
      <bw:content>
         <bw:include src="Mobile_Server_AdminUserList.bml" />
      </bw:content>
   </bw:browser>

   <bw:button id="BU_Cancel" tabindex="11">
      <bw:pos aox="1000" aoy="1000" acx="1000" acy="1000"
                        eox="-414" eoy="-32" ecx="-214" ecy="-8"/>
      <bw:content><bw:t>~Neuen Benutzer erfassen</bw:t></bw:content>
      <bw:events>
         <bw:event method="runProcess( 'PrcEditUser')" name="validate" />
      </bw:events>
   </bw:button>
   <bw:button id="BU_Cancel" tabindex="11">
      <bw:pos aox="1000" aoy="1000" acx="1000" acy="1000"
                        eox="-210" eoy="-32" ecx="-10" ecy="-8"/>
      <bw:content><bw:t>~Prozess verlassen</bw:t></bw:content>
      <bw:events>
         <bw:event  name="clicked" method="{event('B_CAN')}" />
      </bw:events>
   </bw:button>
  </bw:compound>

</bw:portlet>
```

Das Makro (Regeln) für das Element `<bw:dlgrow>` ist wie folgt definiert:

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:rule element="bw:dlgrow">
   <bw:var name="w" expr="iff( @wid eq '', $defw, @wid)" />
   <bw:var name="m" expr="iff( @mtop eq '', $mtop, @mtop)" />
   <bw:var name="tab" expr="$tab+1" />
   <bw:compound id="CP_{@id}" tabindex="{$tab}">
       <bw:pos rel="yflow" asmode="height" aox="0" aoy="0" acx="1000" acy="0"
              eox="{iff( @indent ne '', @indent, 0)}" eoy="{$m}"
              ecx="-{iff( @mright ne '', @mright, 0)}" ecy="1" />

      <bw:if expr="@type ne 'cb'">
         <bw:label id="LA_{@id}"
               for="{iff( @label ne '', concat( substring( @type, 0, 2), '_', @id),'')}"
               align="{$alg}">
            <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="0" eoy="0"
                ecx="{$mleft}" ecy="21" />
           <bw:style><bw:font>Tahoma,10,regular</bw:font></bw:style>
            <bw:content><bw:t><bw:var expr="@label" /></bw:t></bw:content>
         </bw:label>
      </bw:if>

      <bw:if expr="@type eq 'cb'">
         <bw:checkbox id="TE_{@id}" tabindex="1" datasrc="{@datasrc}">
            <bw:pos aox="0" aoy="0" acx="0" acy="0"
               eox="{$mleft+5}" eoy="0" ecx="{$w+$mleft+5}" ecy="21" />
            <bw:style><bw:font>Tahoma,10,regular</bw:font></bw:style>
             <bw:content><bw:t><bw:var expr="@label" /></bw:t></bw:content>
         </bw:checkbox>
      </bw:if>
      <bw:if expr="@type eq 'te'">
         <bw:editor id="TE_{@id}" tabindex="1" wrap="true" borderref="blueborder"
               datasrc="{@datasrc}">
            <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="{$mleft+5}" eoy="0"
               ecx="{$w+$mleft+5}" ecy="{@height}" />
            <bw:style><bw:font>Tahoma,10,bold</bw:font></bw:style>
         </bw:editor>
      </bw:if>
      <bw:if expr="@type eq 'tf'">
         <bw:entry id="TF_{@id}" tabindex="1" align="{iff( @align eq '', 'left', 'right')}"
               format="{@format}">
            <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="{$mleft+5}" eoy="0"
               ecx="{$w+$mleft+5}" ecy="21" />
            <bw:style><bw:font>Tahoma,10,bold</bw:font></bw:style>
            <bw:events>
               <bw:event method="{concat( 'handleEvent( valueChanged,' , @pid, ',' ,
                  concat( '@id', ',' , '@val' , ')' ))}" name="validate" />
            </bw:events>
         </bw:entry>
      </bw:if>
      <bw:if expr="@type eq 'ch'">
         <bw:choice id="CH_{@id}" tabindex="1" datasrc="">
            <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="{$mleft+5}" eoy="0"
               ecx="{$w+$mleft+5}" ecy="21" />
            <bw:style><bw:font>Tahoma,10,bold</bw:font></bw:style>
            <bw:content><bw:iarray><bw:var expr="@val" /></bw:iarray></bw:content>
         </bw:choice>
      </bw:if>
      <bw:if expr="@type eq 'tftf'">
         <bw:entry id="TF_{@id}" tabindex="1" align="{iff( @align eq '', 'left', 'right')}"
               format="{@format}" datasrc="{@datasrc}">
            <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="{$mleft+5}" eoy="0"
               ecx="{$w+$mleft+5}" ecy="21" />
            <bw:style><bw:font>Tahoma,10,bold</bw:font></bw:style>
            <bw:content><bw:var expr="@val1" /></bw:content>
         </bw:entry>
         <bw:label id="LA_{@id}_2" for="TF_{@id}_2" align="{$alg}">
             <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="{$w+$mleft+10}" eoy="0"
               ecx="{$mleft+$defw-$w-5}" ecy="21" />
```



© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
            <bw:style><bw:font>Tahoma,10,regular</bw:font></bw:style>
             <bw:content><bw:t><bw:var expr="@label2" /></bw:t></bw:content>
         </bw:label>
         <bw:entry id="TF_{@id}_2" tabindex="2" align="{iff( @align2 eq '', 'left',
               'right')}" format="{@format2}" datasrc="{@datasrc2}">
            <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="{$mleft+$defw-$w}" eoy="0"
               ecx="{$mleft+$defw+5}" ecy="21" />
            <bw:style><bw:font>Tahoma,10,bold</bw:font></bw:style>
            <bw:content><bw:var expr="@val2" /></bw:content>
         </bw:entry>
      </bw:if>
      <bw:if expr="@type eq 'chch'">
         <bw:choice id="CH_{@id}" tabindex="1" align="{iff( @align eq '', 'left',
               'right')}" format="{@format}" datasrc="">
            <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="{$mleft+5}" eoy="0"
               ecx="{$mleft+@dx+2}" ecy="21" />
            <bw:style><bw:font>Tahoma,10,bold</bw:font></bw:style>
            <bw:content><bw:iarray><bw:var expr="@val1" /></bw:iarray></bw:content>
         </bw:choice>
         <bw:label id="LA_{@id}_2" for="CH_{@id}_2" align="{$alg}">
             <bw:pos aox="0" aoy="0" acx="1000" acy="0" eox="{$mleft+@dx+7}" eoy="0"
               ecx="-{@dx+5}" ecy="21" />
            <bw:style><bw:font>Tahoma,10,regular</bw:font></bw:style>
             <bw:content><bw:t><bw:var expr="@label2" /></bw:t></bw:content>
         </bw:label>
         <bw:choice id="CH_{@id}_2" tabindex="2" align="{iff( @align eq '', 'left',
               'right')}" format="{@format}" datasrc="">
            <bw:pos aox="1000" aoy="0" acx="1000" acy="0" eox="-{@dx}" eoy="0" ecx="0"
               ecy="21" />
            <bw:style><bw:font>Tahoma,10,bold</bw:font></bw:style>
            <bw:content><bw:iarray><bw:var expr="@val2" /></bw:iarray></bw:content>
         </bw:choice>
      </bw:if>
      <bw:if expr="@info ne ''">
         <bw:text align="MiddleLeft">
             <bw:pos aox="0" aoy="0" acx="1000" acy="0" eox="{$w+$mleft+10}"
               eoy="0" ecx="-10" ecy="21" />
            <bw:style>
               <bw:font>Tahoma,10,regular</bw:font>
               <bw:color>#999999</bw:color>
            </bw:style>
             <bw:content><bw:t><bw:var expr="@info" /></bw:t></bw:content>
         </bw:text>
      </bw:if>
   </bw:compound>
</bw:rule>
```

BML ist in seiner Grundform eine XML-Auszeichnungssprache, die vom XML-Transformer ausgewertet und in ein spezifisches Zielformat überführt wird. Dabei wird, vom DialogManager gesteuert, die BML-Beschreibung verwendet, um die darin enthaltenen Befehle auszuführen, die Dialog-Elemente (z.B. Eingabfeld) mit den Entitäten des Business Object Models (BOMs, z.B. „Partner.Nachname“) zu verknüpfen bzw. deren Inhalte in den Dialog zu integrieren und den daraus entstandenen, konkreten Dialog in das Zielformat (z.B. DHTML) zu transformieren.

Auf Grund der flexiblen Transformation können unterschiedliche Zielsysteme unterstützt werden (XML, DHTML, PDF, XUL, XAML etc.). Aktuell unterstützt die Version 4.03 die Zielformate:

© Copyright BISS GmbH, 2001-2012 All Rights reserved

- XML (SmartClient)
- DHTML (BrowserClient)
- PDF (Adobe Reader)

Die folgende schematische Darstellung veranschaulicht den Ablauf der Transformation und die Integration der im DOM vorhandenen Daten sowie die Transformation in die unterschiedlichen Zielformate.

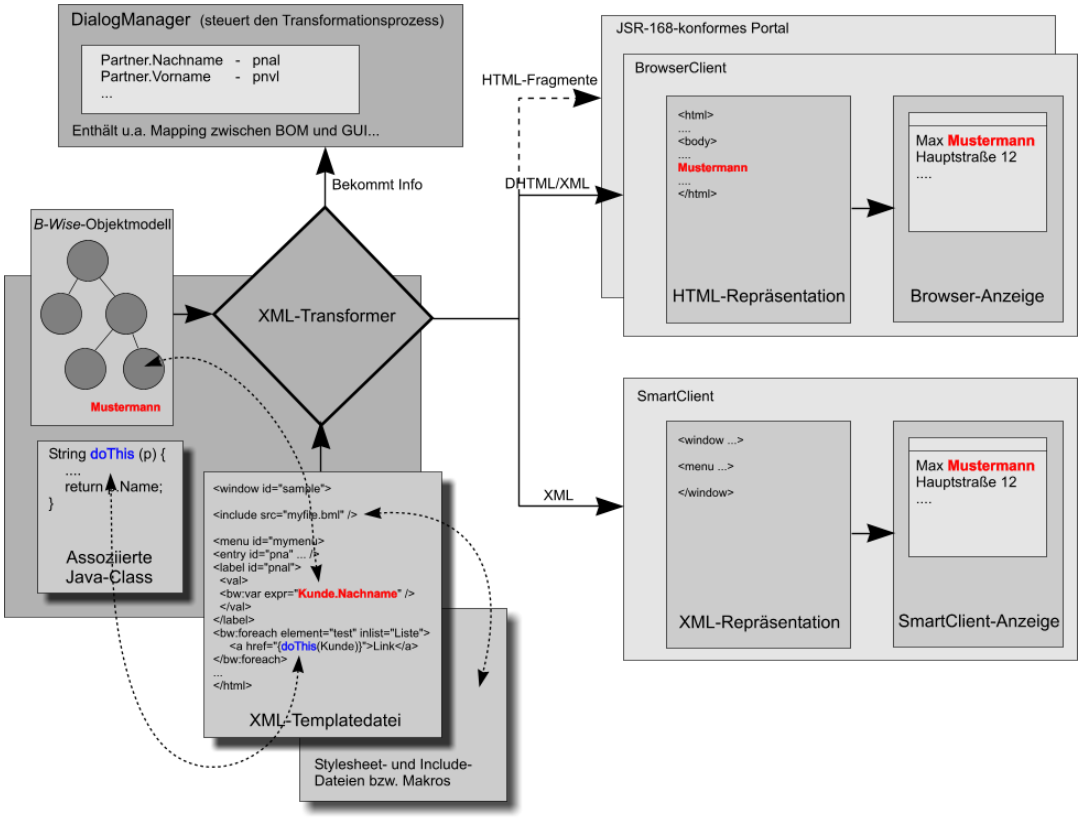



**Abb. 1: Schematische Darstellung der BML-Verwendung**

Um die im Objektmodell enthaltene Information (z.B. Kunden- bzw. Vertragsdaten, in der Abbildung beispielhaft der Nachname "Mustermann") anzeigen zu können, muss diese in das jeweilige Zielformat (z.B. DHTML) gewandelt und an den *B-Wise* Client (in diesem Falle Browser) übergeben werden. Diese Wandlung erfolgt in folgenden Schritten:

1. Auswahl der zugehörigen BML-Definitionsdatei
2. Transformation (Wandlung der BML-Templatedatei in eine DHTML-Datei mit Ersetzung der Variablen sowie Ausführung der BML-Statements)


© Copyright BISS GmbH, 2001-2012 All Rights reserved

3. Parsen der DHTML-Datei (inkl. der Ausführung vorhandener JavaScript-Programme) sowie Anzeige im *B-Wise*-Browser

Neben dem Einsatz zur Dialog- bzw. Dokumentdefiniton dient BML auch innerhalb von XHTML als Mittel zur Verbindung von Anzeigeseite und Objektmodell. Die nachfolgende Dokumentation enthält hierzu zahlreiche Beispiele. Wird BML innerhalb von XHTML verwendet, sind die entsprechenden Elemente (im Folgenden auch häufigs „Tags“ genannt) innerhalb eines eigenen Namensraums anzusiedeln. Dies erfolgt normalerweise mit der folgenden Namensraum-Deklaration innerhalb des `<html>`-Tags:

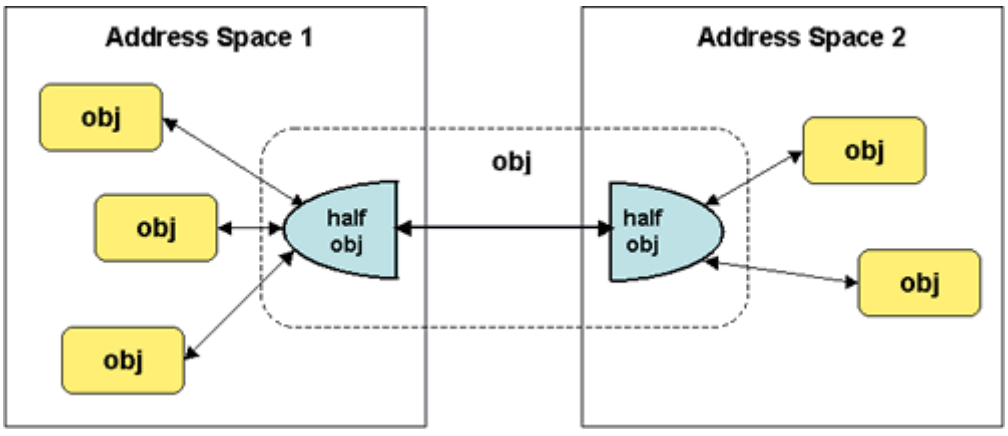
```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
      xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">
```

Für die dynamische Verbindung zwischen *B-Wise* Server und dem im Client dargestellten Dialog (also z.B. das dynamsche Setzen von Werten oder Ändern von Darstellungs-Styles) wird das sog. „Half-Object Pattern“ verwendet.

Das "Half Object Protocol" wurde von Gerard Meszaros in "Pattern Languages of Program Design" von James Coplien und Douglas Schmidt veröffentlicht (Addison Wesley 1995, ISBN 0-201-60734-4).

Das „Half-Object Pattern“ ist für die transparente Verteilung eines Objekts über mehrere Adressräume hinweg gedacht. Hierzu wird ein Objekt in mehrere Subobjekte aufgeteilt. Jedes dieser Subobjekte repräsentiert das Gesamtobjekt in seinem Adressraum und enthält die für den Adressraum relevanten Informationen und Methoden. Die einzelnen Subobjekte wiederum synchronisieren sich mittels Mechanismen der Interprozess-Kommunikation. DIes erfolgt in *B-Wise* mittels XML-RPC oder WebServices (SOAP über http).

Die Abbildung 2 stellt das Prinzip des „Half-Object Patterns“ dar.




© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Abb. 2: Schema des „Half-Object Patterns“**

Abbildung 3 zeigt die Situation zur Laufzeit der Anwendung für einen JavaClient. Im .net Client ist die Presentation Engine in C#, im BrowserClient sind Teile der Presentation Engine in JavaScript implementiert.

Die Presentation Logic ist in *B-Wise* innerhalb des sog. „Dialog-Managers“ realisiert, der für die gesamte Steuerung des GUI verantwortlich ist. Teil des Dialog-Managers ist der BML-Transformer, der für die korrekte Umsetzung der in BML definierten Dialoge, Anzeigeseiten und/oder Dokumente zuständig ist.

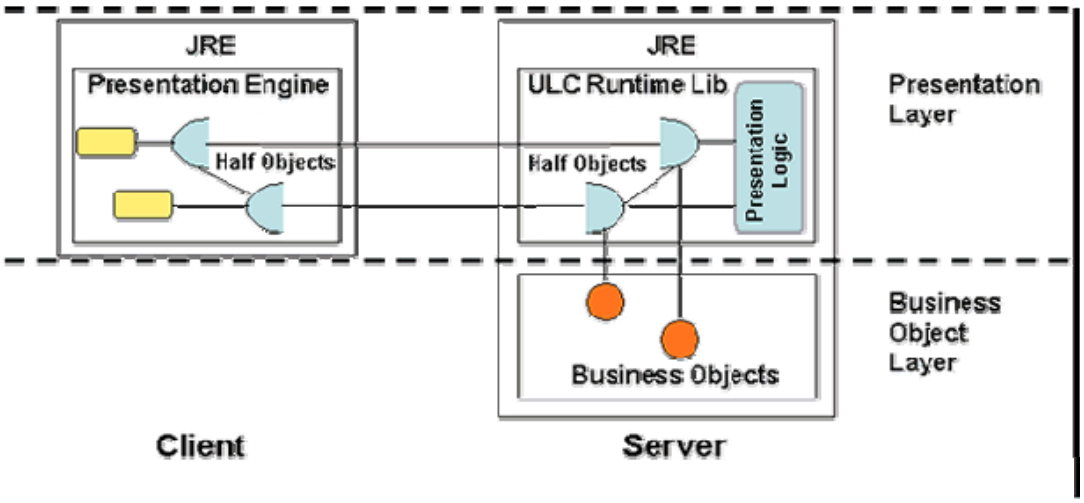



**Abb. 3: „Half-Object-Pattern“ zur Laufzeit**

## 1.2 Warum BML ?

BML wurde nach einer eingehenden Evaluation vergleichbarer Techniken und der verfügbaren Frameworks entwickelt. Gegenstand der Analyse waren insbesondere

- Java Server Pages (JSP)
- Java Server Faces (JSF)
- XML/XSLT-Transformationen
- Frameworks wie Tapestry

An dieser Stelle sollen daher kurz die Vorteile von BML im Vergleich zu den o.g. Alternativen aufgezählt werden.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Erfahrungen haben gezeigt, dass Skripting-Techniken wie JSP dazu führen, dass ca. 40% – 50% einer Seite aus Code bestehen, der in großen Teilen Logiken enthält, welche aus Architektursicht in die Logic-Tier gehören. Darüberhinaus können derartige Seiten nicht mit handelsüblichen Editoren bearbeitet werden.

In gut aufgestellten Projekten sind unterschiedliche Teammitglieder mit spezifischen Fähigkeiten für jeweils spezielle Aufgabengebiete verantwortlich. So sollte das Dialogdesign nicht nur architektonisch von der Prozess- bzw. Logikentwicklung getrennt, sondern auch an unterschiedlich ausgebildete Teammitglieder vergeben werden. Dies ist umso leichter, je besser die verwendeten Techniken eine derartige Arbeitsaufteilung unterstützen.

BML konzentriert sich ausschließlich auf die Belange der Präsentation und ist über klar definiert Schnittstellen mit der zugehörigen Applikationslogik verknüpft. BML hat nur minimalen Einfluss auf die Implementierung der zugehörigen XHTML-Seite (bzw. des BWD-Dokuments), so dass Dialog/Dokument-Designer bzw. –Implementatoren eine optimale Unterstützung ohne unnötige Störungen erfahren. BML setzt keine spezifischen Konstrukte voraus, so dass sowohl „legacy" HTML-Code als auch strukturierter XHTML/CSS-Code optimal unterstützt werden. BML erfordert keine Java-Programmierkenntnisse und erlaubt dem Designer, in seiner Domain zu arbeiten.

BML verfügt über Konstrukte, die eine effiziente Unterstützung von Prototypen erlauben; diese Prototypen können anschließend durch einfaches „Anreichern" mit XML-Tags in produktive Seiten überführt werden, ohne den eigentlichen (XHTML/CSS-) Design-Code zu ändern oder gar wegzuwerfen.

BML ist (zusammen mit weitere *B-Wise*-Komponenten) daher am ehesten mit Frameworks wie Tapestry vergleichbar, hat gegenüber diesen jedoch eine Reihe von Vorteilen.

BML beschränkt sich nicht nur auf die Verbindung von Präsentation und Applikationslogik. Eine wesentliche Eigenschaft von BML ist die Unterstützung des technischen Dialogdesigns bzw. der Strukturierung der entsprechenden Seiten. Hierzu unterstützt BML sowohl Ableitungs- als auch Aggregationsstrukturen, so dass störende Redundanzen vermieden werden und warbarer Code entstehen kann.

Schließlich dienen BML-Konstrukte der „Localization" (Nationalisierung) der Präsentationsschicht, also der Implementierung von Anzeige- und Dialogseiten sowie Dokumenten in unterschiedlichen Sprache ohne die Schaffung von Redundanzen.

Durch die Strukturunterstützung und die Localization-Fähigkeiten geht BML daher auch weit über die Fähigkeiten von XML/XSLT-Transformationen hinaus. Ein weiterer grundlegender technischer Unterschied zwischen XML/XSLT einerseits und BML andererseits

© Copyright BISS GmbH, 2001-2012 All Rights reserved

besteht darin, dass XSLT das XML-Stylesheet vor dem Einlesen der XML-Input-Dateien kompiliert, während BML verschränkt kompiliert und einliest, wobei der Input alternativ aus einem beliebigen Java-DOM stammen kann. Weitere Unterschiede zwischen XML/XSLT und BML enthält die folgende Tabelle.

| Eigenschaft | XML/XSLT | BML |
|---|---|---|
| include beliebiger XML-Fragmente | Nur auf Toplevel möglich | An beliebieger Stelle möglich |
| include abhängig von Bedingungen / Variablen | Nein | Ja |
| Inheritance | Nein | Ja |
| Werte-Änderung von Variablen / Unterstützung von Schleifen-abhängigen-Variablen | Deklaration nur auf TopLevel-Template Ebene möglich | Ja |
| Zuweisung von beliebigen Ausdrücken an Variablen | Nein | Ja |
| Dynamische Variablennamen | Nein | Ja |
| Dynamische XPath-Expressions (mit Variablen) / Meta-Evaluation (eval-Methode) | Nein | Ja |
| Alternativer Zugriff auf XML-Input und entsprechendes Java-DOM | Nein | Ja |
| Schleifen-Beschränkung (for-each) auf die ersten n Elemente | Nein | Ja |
| DOM-Zugriffe mit beliebiger Navigationsrichtung (Parent/Sibling/Child/..) | Ja | Nur Child-Richtung |
| Formatierfunktionen für Date/Timestamp | Nein | Ja |
| Zugriff auf Java-Script-Methoden innerhalb von Expressions | Nein | Ja |
| Expressions mit dynamisch erweiterbarem Funktionsvorrat (z.B. Partner.fullName()) | Nein | Ja |
| Übernahme von Entities (`<bw:!DOCTYPE ... />`) in die Ausgabe-Datei | Nein | Ja |
| Eingebaute Transformations-Regeln (nach Benutzer-Anforderungen) | Nein | Ja |
| Beliebige Erweiterbarkeit | Nein | Ja |

Schließlich haben Messungen ergeben, dass XML-Transformationen ca. um Faktor 3-5 performanter ablaufen als vergleichbare XSLT-Transformationen.

Im Folgenden nochmals die Zusammenfassung der wichtigsten BML-Vorteile:


© Copyright BISS GmbH, 2001-2012 All Rights reserved

- Gezielte Unterstützung der Präsentationsschicht
- Einfache Verwendung, daher schnell zu erlernen
- Unterstützung der Seiten-Strukturierung (Ableitungen und Aggregationen)
- Einfache und effiziente Localization-Unterstützung
- Mächtige Sprachkonstrukte
- BML geht weit über eine einfache Transformation hinaus und definiert Client-übergreifende Widget-Konstrukte mit zahlreichen Smart-Client- bzw. Browser-optimierten Funktionen
- Unterstützung von Teamentwicklung
- Hervorragende Performance, damit breites Einsatzfeld
- Beliebige Erweiterbarkeit

## 1.3 Entwicklungsmethodik

BML entstand vor dem Hintergrund einer definierten Entwicklungsmethodik. Diese zielt auf eine effiziente, schnelle und einfache Implementierung von *B-Wise* Anzeigeseiten, -Dialogen sowie -Dokumenten mit möglichst früher und intensiver Integration des Kunden (Endbenutzers).

Benutzerschnittstellen sind Bereiche eines Software-Systems, die großen Einfluss auf die Nutzbarkeit des Software-Produkts und somit auf dessen Akzeptanz haben. Darüber hinaus stellen sie Teile des Systems dar, die im Projektverlauf häufigen Änderungen unterworfen sind. Hier ist auch der Diskussionsbedarf mit dem Kunden bzw. dem Endbenutzer am größten. Die durch BML unterstützte Entwicklungsmethodik bietet dabei folgende Vorteile:

- Vollständige Entkoppelung des UI-Designprozesses vom übrigen Entwicklungsprozess (dadurch können Anzeigeseiten von Design- bzw. UI-Spezialisten entworfen werden, ohne den übrigen Entwicklungsprozess zu stören) - die Schnittstelle bildet das Repository
- Trennung von Implementierung und Design; die visuelle Erscheinung der Seiten (Basislayout, Farben, Fonts, statische Texte, etc.) können von der Codeimplementierung getrennt werden
- Erstellung von Prototypen, die nahtlos in produktiven Code überführt werden können; damit wird eine frühzeitige Einbindung des Endkunden möglich
- Iterativer Entwicklungsprozess; die in Code gegossenen produktiven Anzeigeseiten (BML-Templatedateien) können prototypisch weiterentwickelt werden (produktiver Code und prototypische Anteile können beliebig gemischt werden)
- Prototypische Anteile können auch zur Laufzeit im produktiven Code vorkommen; damit lassen sich in einer laufenden Applikation prototypisch neue Funktionen integrieren (Rapid Prototyping bzw. Expreme Programming)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Kapitel 2
# BML Tutorial

Dieser Abschnitt beschreibt zwei einfache, kurze (Beispiel-) BML Dialogdateien sowie die Schritte, die erforderlich sind, um diese zu erstellen. Dabei handelt es sich um eine

- Dialogseite (interaktiver Dialog mit GUI-Controls) sowie eine
- Anzeigeseite (XHTML in Kombination mit BML-Tags)

Anhand dieser Beispiele soll die Arbeitsweise von BML erläutert sowie kurz die erforderlichen Werkzeuge (Tools) und deren Handhabung vorgestellt werden.

## 2.1 *B-Wise* Benutzerschnittstelle

Bevor wir konkrete Dialoge bzw. Anzeigeseiten erstellen, ist es sinnvoll, kurz die Philosophie der *B-Wise* Benutzerschnittstelle zu erläutern. Eine ausführliche Erläuterung der *B-Wise* Bedienphilosophie ist in einem separaten Dokument enthalten. Wir wollen uns hier daher auf das Layout bzw. die funktionalen Aspekte der Benutzerschnittstelle (GUI - Graphical User Interface) konzentrieren.

*B-Wise* verwendet - unabhängig vom verwendeten Client - zur Präsentation von Informationen oder Vorgängen/Prozessen das Portal-Konzept. Ein Portal ist ein (normalerweise rechteckiger) Bildschirmbereich, in dem Informationen mit Hilfe sog. Portlets dargestellt werden. Portlets sind eigenständige Fenster, die im Portal mehr oder minder fest eingebettet sind. Die folgende Abbildung (Abb. 4) zeigt dieses Konzept schematisch und die Abbildung 5 enthält ein Beispiel für ein konkretes Portal mit unterschiedlichen Portlets.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

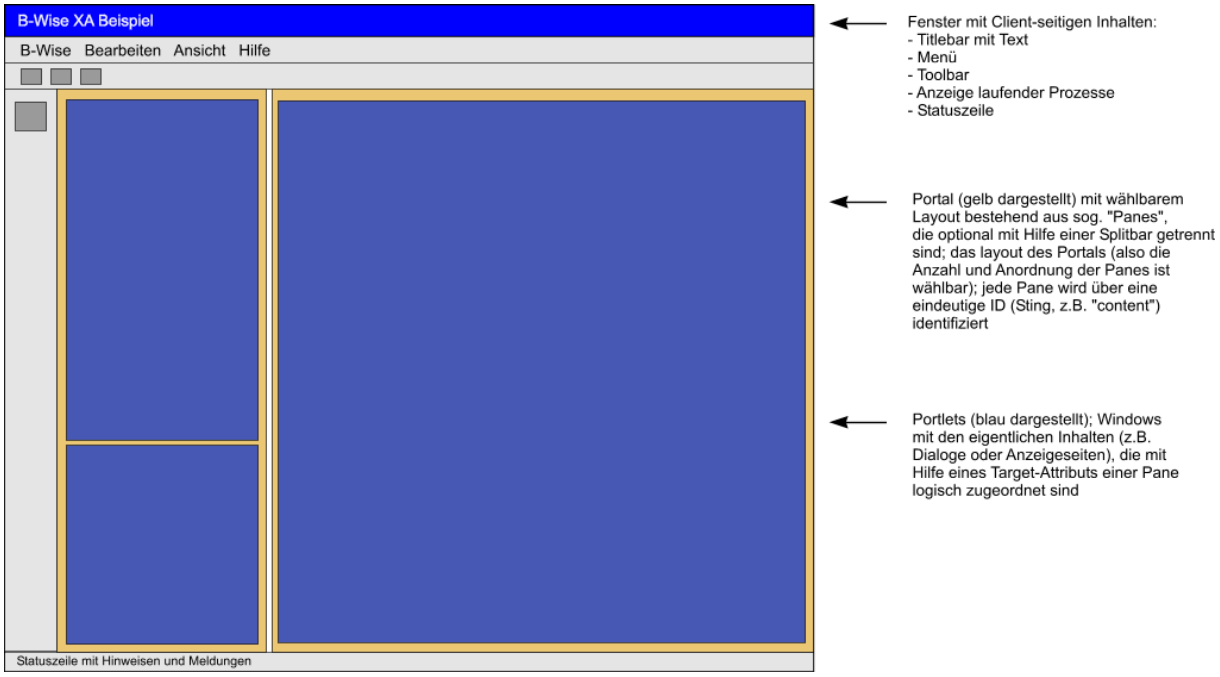

B-Wise XA Beispiel
B-Wise Bearbeiten Ansicht Hilfe
Statuszeile mit Hinweisen und Meldungen
Fenster mit Client-seitigen Inhalten:
- Titlebar mit Text
- Menü
- Toolbar
- Anzeige laufender Prozesse
- Statuszeile
Portal (gelb dargestellt) mit wählbarem Layout bestehend aus sog. "Panes", die optional mit Hilfe einer Splitbar getrennt sind; das layout des Portals (also die Anzahl und Anordnung der Panes ist wählbar); jede Pane wird über eine eindeutige ID (Sting, z.B. "content") identifiziert
Portlets (blau dargestellt); Windows mit den eigentlichen Inhalten (z.B. Dialoge oder Anzeigeseiten), die mit Hilfe eines Target-Attributs einer Pane logisch zugeordnet sind

**Abb. 4: Schematische Darstellung des Portal-Konzeptes**

Jedes Portlet (in Abb. 4 blau dargestellt) zeigt einen spezifischen Inhalt. Portlets können sowohl vom Prozess gesteuert als auch vom Anwender innerhalb des Portals platziert werden. Hierzu definiert das Portal unterschiedliche Panes, also Bereiche, in denen Portlets platziert werden können. Die Panes sind in Abb. 4 gelb dargestellt. Zwischen zwei Panes kann optional eine sog. „Splitbar“ eingefügt werden, mit deren Hilfe die Breiten/Höhen der angrenzenden Panes (und damit auch der darin enthaltenen Portlets) vom Anwender verändert werden können. Das Portal selbst ist ebenfalls in BML definiert.

Zusätzlich zur festen Platzierung innerhalb des Basisportals können Portlets losgelöst innerhalb eines frei beweglichen Fensters angezeigt werden. Dieses Fenster kann dabei sowohl modal als auch nicht-modal sein. Hierzu erzeugt der Prozess ein Dialogfenster, dessen Parameter aus einer Definitionsdatei gelesen werden. Innerhalb des Fensters wird wiederum ein Portal dargestellt, innerhalb dessen die Portlets angezeigt werden können. Die Parameterdatei sieht wie folgt aus (Beispiel):

```
<window id="wnd0" title="Titel des Fensters">
  <decoration sizeable="false" />
  <geometry>
     <centered width="450" height="150" />
  </geometry>
</window>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Die Inhalte sind weitgehend selbsterklärend. Der `title` erscheint innerhalb der Titlebar des Fensters, die Attribute des `<decoration>`-Elements definieren u.a. die Art des Fenster-Randes (im vorliegenden Fall eine sog. „sizeborder“, d.h. ein Rand, der eine Größenänderung des Fensters ermöglicht); das `<geometry>`-Element bzw. dessen Subelemente definieren die Größe und Position des Fensters (in vorliegenden Beispiel zentriert über dem Hauptfenster sowie 450 Pixel breit und 150 Pixel hoch).

In vielen Fällen ist einem Portlet auch ein Prozess zugeordnet (besser gesagt: einem Prozess ist ein Portlet zugeordnet). Manchmal kann ein Prozess jedoch auch mehrere Portlets gleichzeitig verwenden, um unterschiedliche Informationen anzuzeigen.

Befinden sich mehrere Portlets im Portal, so gibt es immer ein „aktives“ Portlet; das ist dasjenige, welches die Tastatureingaben entgegen nimmt. Das aktive Portlet wird dem Anwender optisch z.B. durch die Farbe der Titlebar oder die Farbe des Titlebar-Textes signalisiert.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

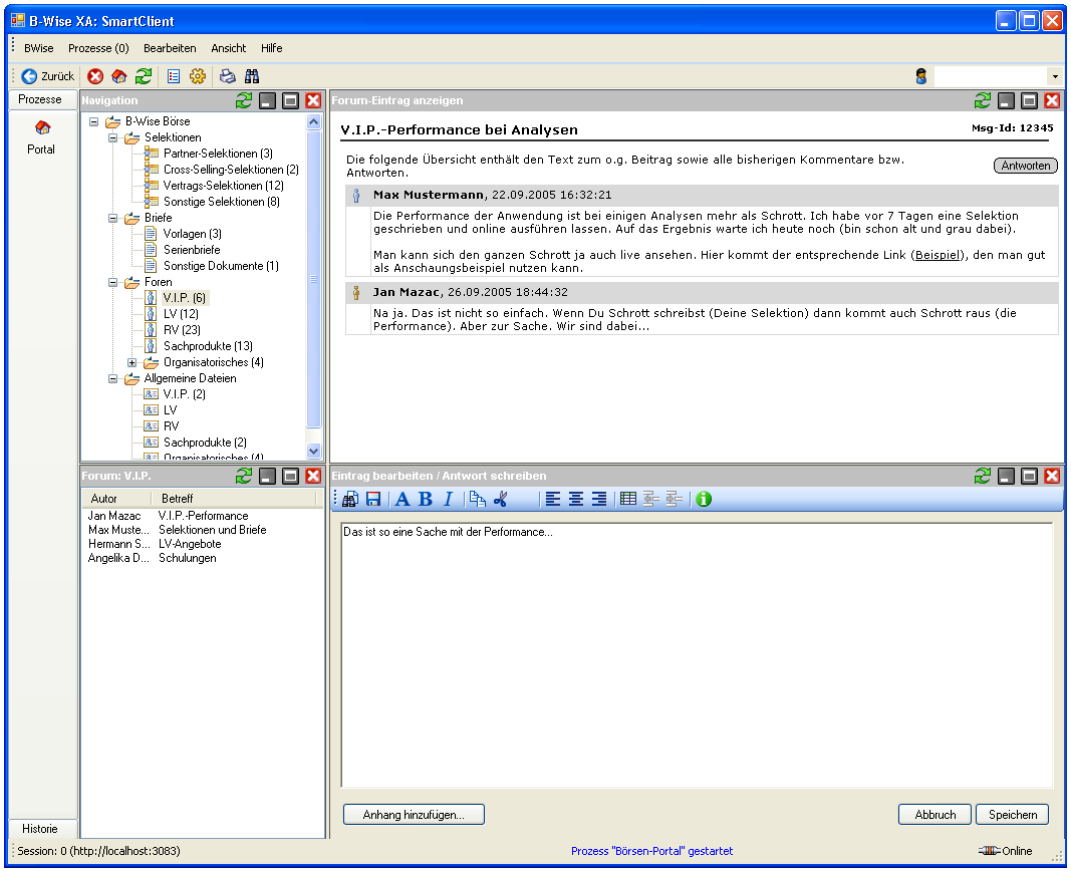

B-Wise XA: SmartClient
BWise Prozesse (0) Bearbeiten Ansicht Hilfe
Zurück
Prozesse
Portal
Historie
Navigation
B-Wise Börse
Selektionen
Partner-Selektionen (3)
Cross-Selling-Selektionen (2)
Vertrags-Selektionen (12)
Sonstige Selektionen (8)
Briefe
Vorlagen (3)
Serienbriefe
Sonstige Dokumente (1)
Foren
V.I.P. (6)
LV (12)
RV (23)
Sachprodukte (13)
Organisatorisches (4)
Allgemeine Dateien
V.I.P. (2)
LV
RV
Sachprodukte (2)
Forum-Eintrag anzeigen
V.I.P.-Performance bei Analysen
Msg-Id: 12345
Die folgende Übersicht enthält den Text zum o.g. Beitrag sowie alle bisherigen Kommentare bzw. Antworten.
Antworten
Max Mustermann, 22.09.2005 16:32:21
Die Performance der Anwendung ist bei einigen Analysen mehr als Schrott. Ich habe vor 7 Tagen eine Selektion geschrieben und online ausführen lassen. Auf das Ergebnis warte ich heute noch (bin schon alt und grau dabei).
Man kann sich den ganzen Schrott ja auch live ansehen. Hier kommt der entsprechende Link (Beispiel), den man gut als Anschaungsbeispiel nutzen kann.
Jan Mazac, 26.09.2005 18:44:32
Na ja. Das ist nicht so einfach. Wenn Du Schrott schreibst (Deine Selektion) dann kommt auch Schrott raus (die Performance). Aber zur Sache. Wir sind dabei...
Forum: V.I.P.
Autor
Betreff
Jan Mazac V.I.P.-Performance
Max Muste... Selektionen und Briefe
Hermann S... LV-Angebote
Angelika D... Schulungen
Eintrag bearbeiten / Antwort schreiben
Das ist so eine Sache mit der Performance...
Anhang hinzufügen...
Abbruch
Speichern
Session: 0 (http://localhost:3083)
Prozess "Börsen-Portal" gestartet
Online

**Abb. 5: *B-Wise* Portal mit Portlets (SmartClient)**

Nach dieser kurzen Vorstellung des Portal-Konzeptes können wir nun die zugehörigen Dialoge bzw. Anzeigeseiten definieren, die - gemäß dem oben Gesagten - natürlich als Portlets realisiert sind.

## 2.2 Dialogseite

Die Dialogseite (ein Portlet) enthält die gewünschten Eingabe-Elemente wie Entryfields, Buttons oder Choices und wird normalerweise mit Hilfe des *B-Wise* IDE Designers erstellt. Der Designer arbeitet in zwei Modi:

- Design-Modus
- Text-Modus

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Im *Design-Modus* können die verfügbaren Elemente interaktiv, grafisch im Dialog angeordnet und deren Eigenschaften mit Hilfe entsprechender Editoren definiert werden. Der Designer mit dem Beispiel-Portlet ist in Abbildung 6 dargestellt.

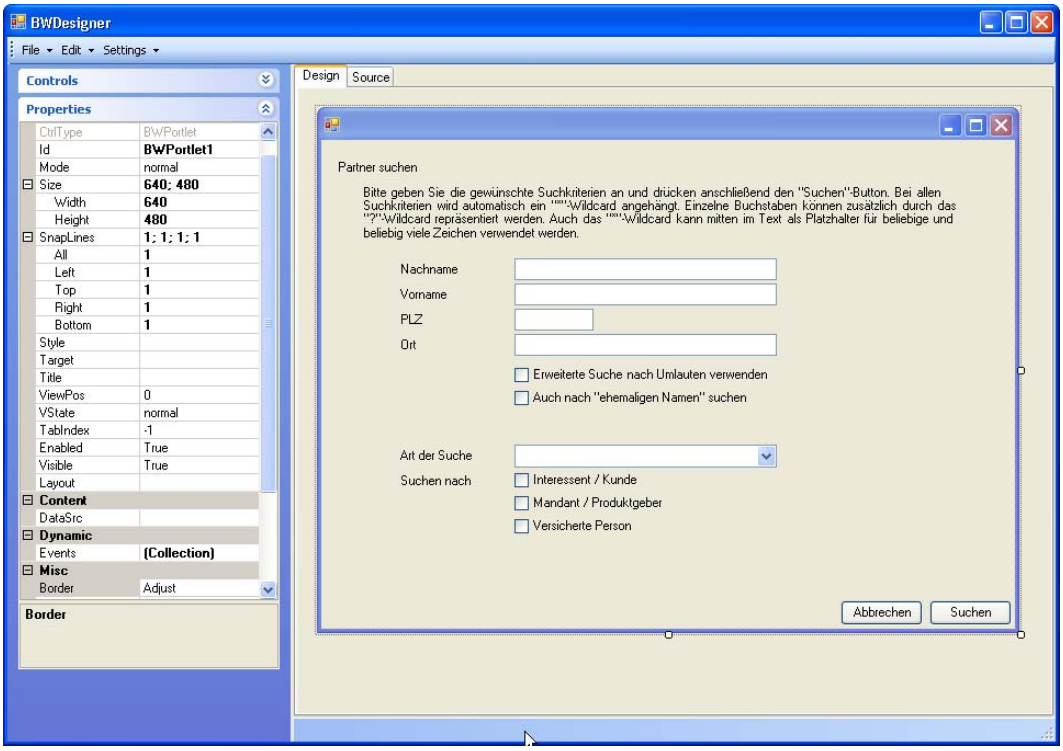



**Abb. 6: *B-Wise* IDE Designer (Design-Modus)**

Sollen komplexe Strukturen (z.B. bedingte Dialoganteile oder dynamisch sich wiederholende Dialogelemente) implementiert werden, steht der Text-Modus (auch Programmier-Modus genannt) zur Verfügung. Hier kann die BML-Definition textuell bearbeitet werden[1] (siehe auch Abbildung 7).

Bis auf einfache Dialoge sind viele Dialoge einer B-Wise Anwendung recht komplex. Dies insbesondere, um Redundanzen zu vermeiden oder die klare Architektur von B-Wise zu unterstützen. Häufig erfolgt daher die Bearbeitung der Dialoge mit Hilfe des Design-Modus als Fragmente, die dann im Text-Modus zusammengestellt werden.

[1] Selbstverständlich kann die Portlet-Definition (XML-Datei) auch mit Hilfe eines beliebigen Editors bearbeitet werden. Hierzu empfiehlt sich ein Editor, der zumindest die XML-Datei auf ihre „wohlgeformtheit“ (well formed XML) überprüft.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

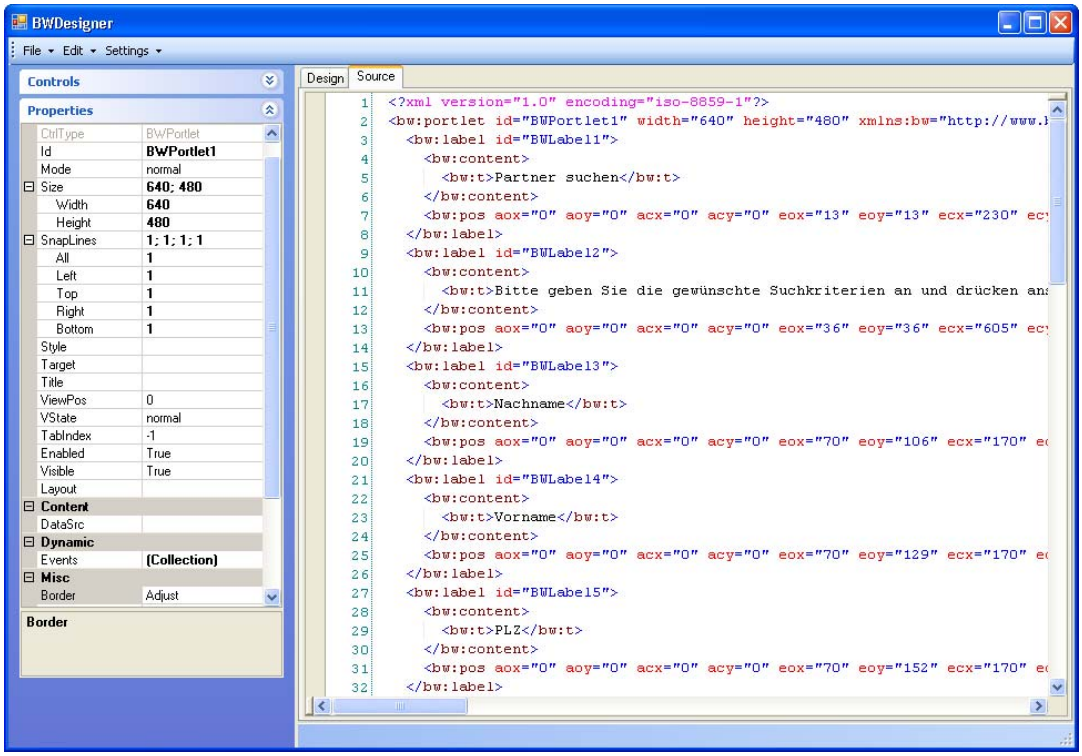



**Abb. 7: *B-Wise* IDE Designer (Text-Modus)**

Ziel des vorliegenden Beispiels ist ein einfacher Suchdialog ohne komplexe Strukturen, der vollständig im Design-Modus erstellt und bearbeitet werden kann. Der Dialog ist innerhalb des Designers in Abb. 6 dargestellt.

Um die gewünschte Dialogseite zu erstellen, verwenden wir zunächst den Design-Modus und definieren alle erforderlichen Elemente. Diese wählen wir aus dem linken Bereich des Designers, der mit „Controls“ betitelt ist. Ein Element wird durch einen einfachen Click ausgewählt und anschließend mit Hilfe eines weiteren Clicks auf der Portlet-Fläche platziert. In der aktuellen Version sind folgende Elemente verfügbar (siehe Abb. 8).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

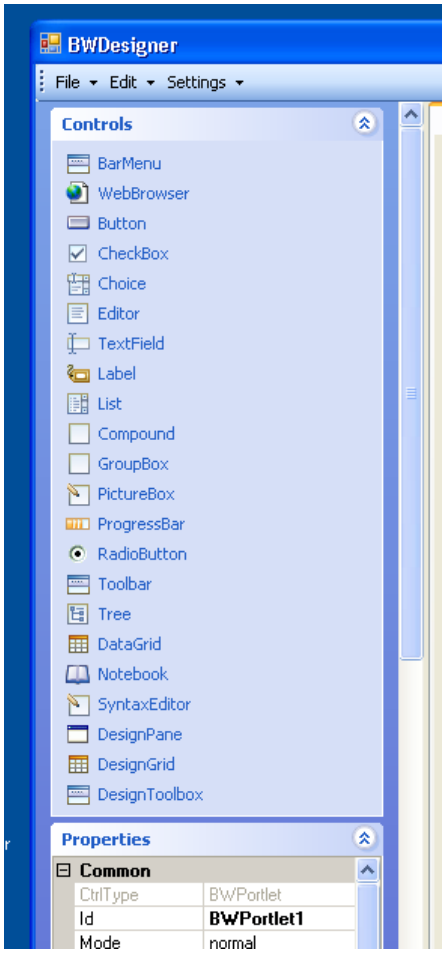



**Abb. 8: Verfügbare Control-Elemente (GUI-Controls)**

Aus der Liste der verfügbaren Dialogelemente wählen wir zunächst das Label und positionieren es auf der Dialogseite. Das Label kann - wenn es selektiert ist - mit Hilfe der Maus verschoben werden. Auch die Größe des sog. „bounding rectangles“ kann mit Hilfe der „handles“ verändert werden. Wir positionieren das Label links oben im Portlet. Anschließend ändern wir den Standard-Text innerhalb des „Property“-Editors (dieser befindet sich ebenfalls auf der linken Seite des Designers unterhalb der Controls und ist mit „Properties“ betitelt). Der Property-Editor zeigt die Eigenschaften des gewählten (selektierten) Elements an. Diese Eigenschaften sind vom Typ des Elements abhängig. Für das Label finden wir recht weit unten den Eintrag „Value“ in der Rubrik „Content“. Hier geben wir den Wert „Partner suchen“ ein.

Aus der Liste der verfügbaren Dialogelemente wählen wir nun ein weiteres Label und positionieren es auf der Dialogseite unterhalb des ersten Labels. Anschließend wird die Größe des „bounding rectangles“ entsprechend vergrößert, so dass sich der Text fast über die gesamte


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Breite des Dialogs erstrecken kann. Innerhalb des Property-Editors wird nun das Scaling derart definiert, dass sich das Label mit der Breite des Dialogs entsprechend in seiner Breite verändert. Hierzu wird der Wert „AnchorStyle2“ der Rubrik „Common“ auf den Wert „Top,Right“ gesetzt. Dies erfolgt mit Hilfe einer grafischen Eingabe (siehe Abb. 9).

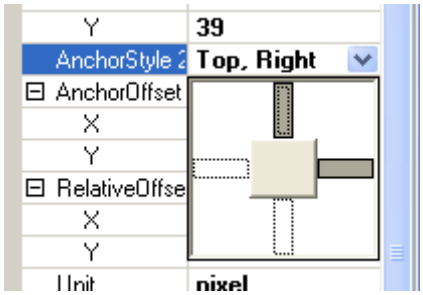



**Abb. 9: Definition des Framings**

Clicken Sie hierzu die Rechtecke derart an, dass sich das oben gezeigte Bild ergibt. Dies definiert den „Anker“ der rechten unteren Ecke des Labels so, dass sich diese mit dem umgebenden Portlet wie folgt ändert: Der Abstand zwischen dem oberen Rand des Portlets und der rechten unteren Ecke bleibt konstant und der Abstand zwischen dem rechten Rand und der rechten unteren Ecke bleibt konstant. Weitere Möglichkeiten zur Definition des Scaling-Verhaltens sind im Abschnitt 9.11 (*B-Wise* Layouter) beschrieben.

Wir können nun den Text des Labels erfassen. Da es sich um einen längeren Text handelt, verwendet wir hierzu den Texteditor des Designers, der sich beim Property „Value“ hinter dem Button mit den drei Punkten verbirgt. Erfassen Sie nun den gewünschten Text und schließen anschließend den Editor; der Text wird automatisch als Wert des Labels übernommen.

Auf diese Weise definieren wir alle weiteren Controls:

- die weiteren Label
- die Eingabefelder (diese heißen „Texfield“)
- die Combobox (diese heißt „Choice“)
- die Checkboxen
- und die Buttons.

Als letztes müssen wir noch die Verbindung (Assoziation) zwischen dem Dialog-Portlet und der gewünschten Pane herstellen. Ohne an dieser Stelle näher darauf einzugehen, verwendet wir die „cont“-Pane. Definieren Sie hierzu im Designer das Portlet selbst (z.B. durch Click auf die Titlebar) und geben für die Property „Target“ der Rubrik „Common“ den Wert „cont“ ein.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Schließlich speichern wir den Dialog in der Datei `Test_SampleDialog.plt` ab.

Wir können den Dialog nun unmittelbar innerhalb von *B-Wise* anzeigen. Hierzu benötigen wir allerdings einen Prozess. Ein Prozess ist ein funktionaler Ablauf, der aus einer logischen Abfolge unterschiedlicher Schritte besteht. In den meisten Fällen spiegeln sich die Schritte in Dialogen oder Anzeigeseiten wider.

Um schnell und einfach effiziente Prototypen erstellen zu können, integriert B-Wise eine Skriptsprache, mit deren Hilfe wir einen einfachen Prozess innerhalb weniger Minuten erstellen können. Bei der Skriptsprache handelt es sich übrigens um Groovy, eine der Java-Sprache sehr ähnliche Skriptsprache, die unter der ID JSR-268 von der Java-Community standardisiert ist.

Ohne auf die Inhalte an dieser Stelle genauer eingehen zu können, erstellen wir mit Hilfe eines Editors den folgenden Code:

```
import com.biss.dlgsys.ifc.PortletDescrs;
import com.biss.process.GUIProcess;

class TestProzess
  extends GUIProcess {

  public TestProzess() {
    super();
  }

  public String getName() {
    return "Einfacher Testprozess";
  }

  public String getPreferredPortal() {
    return "System_Standard.por";
  }

  public PortletDescrs definePortlets() {
    PortletDescrs portlets = super.definePortlets();
    portlets.add( "Test_SampleDialog.plt");
    return portlets;
  }
```



© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  public void initialize() {
    super.initialize();
  }

  public void start() {
    super.start();
  }
}
```

Die relevanten Stellen sind im Code fett dargestellt. Es handelt sich dabei um den Namen des zu verwendenden Portals und um den Namen unseres Portlets.

Das Portal ist im Folgenden dargestellt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bw:portal
  xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml"
  id="BWisePortalBasePage"
  name="BWisePortalBasePage">

  <bw:portletcontainer id="root" class="bw_container"
       px="0" py="0" dx="100%" dy="100%" split="V">
    <bw:portletpane id="navi" class="bw_navi"
      px="0" py="0" dx="240" dy="100%" radio="1" />
    <bw:portletpane id="cont" class="bw_pane"
      px="N" py="0" dx="*" dy="100%" radio="0" />
  </bw:portletcontainer>

</bw:portal>
```

Es handelt sich hier um das *B-Wise* Standardportal, bestehend aus zwei Panes, die durch eine Splitbar getrennt sind.

Ein Portal (`<bw:portal>`) besteht aus Containern, die wiederum Container oder Panes beherbergen können. Eine Pane dient letztlich als Behälter für Portlets. Ein Container besteht immer aus zwei Hälften, die optional durch eine Splitbar getrennt sein können. Die Splitbar kann dabei horizontal oder vertikal im Container dargestellt werden, was den Container dann in zwei übereinander bzw. nebeneinander stehende Hälften teilt. Sowohl Container als auch Panes haben eine „id“ sowie eine Position bzw. Breite und Höher. Eine Pane kann zusätzlich im „radio“-Modus arbeiten; in diesem Modus wird nur eines der darin enthaltenen Portlets maximiert dargestellt.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Innerhalb einer Pane können prinzipiell beliebig viele Portlets enthalten sein bzw. dargestellt werden. Dabei kann ein Portlet eine explizite Höhe bzw. Breite haben oder es kann durch die Pane „layoutet" werden. Im letzteren Fall berechnet die Pane die Höhe bzw. Breite eines Portlets anhand eines definierten Algorithmus und weist diese dem Portlet zu. So wird z.B. ein einzelnes Portlet in der vollen Breite und Höhe der Pane dargestellt. Werden zwei Portlets innerhalb einer Pane angezeigt, so teilen sie sich die Höhe und werden in der vollen Breite dargestellt.

Die „`id`" einer Pane ist das „`Target`" des Portlets. Mit Hilfe dieser beiden Attribute werden Portlet und Pane assoziiert, d.h. ein Portlet innerhalb einer definierten Pane angezeigt. Innerhalb des oben gezeigten Standardportals wird unser neu definiertes Portlet in der „cont"-Pane angezeigt.

Speichern Sie den oben dargestellten Code in einer Datei mit dem Namen „TestProzess.groovy" innerhalb des „groovy"-Verzeichnisses einer *B-Wise* Installation ab.

Starten Sie den *B-Wise* Server und loggen sich mit dem SmartClient ein. Der obige Prozess wird nun in der Liste der verfügbaren Skripte mit dem Namen der Datei („TestProzess") ohne die Dateierweiterung „groovy" angezeigt. Nachdem Sie den Prozess gestartet haben, sieht das zugehörige Bild wie folgt ab (siehe Abbildung 10).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

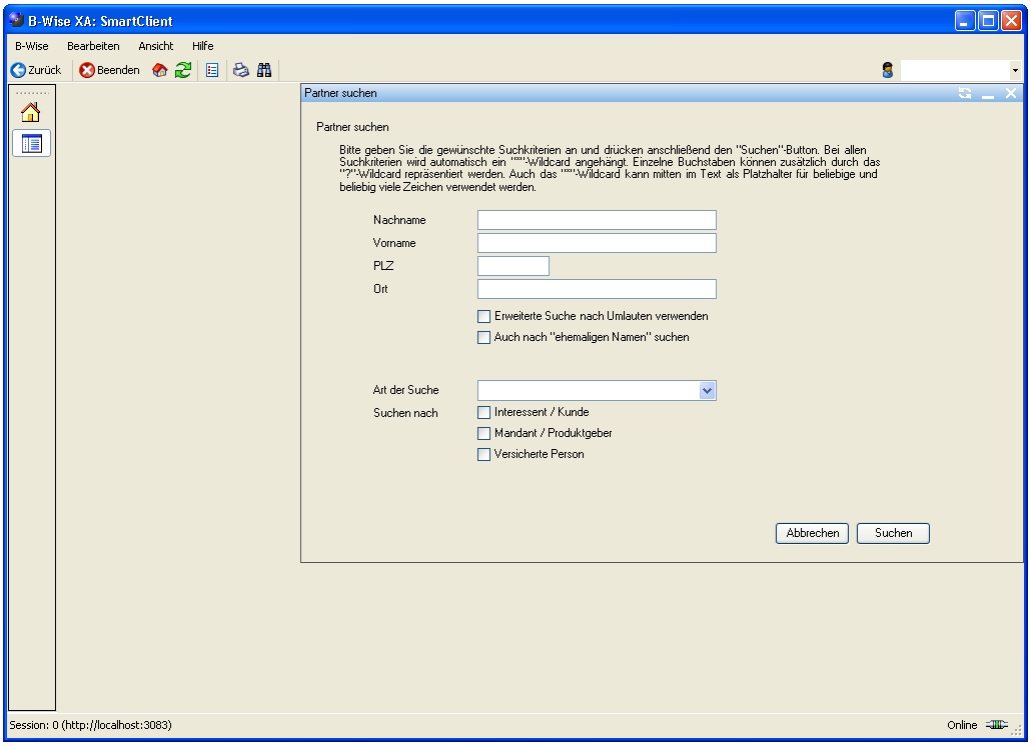



**Abb. 10: Beispiel-Portlet im Rahmen des Test-Prozesses**

Um das Portlet in der vollen Höhe der Pane darzustellen, müssen wir noch die explizit definierten Abmaße ändern oder löschen (diese Werte werden jedesmal vom Designer erzeugt, so dass sie ggf. explizit gelöscht werden müssen). Innerhalb der Portlet-Definitionsdatei („Test_SampleDialog.plt“) befinden sich diese Angaben zu Beginn der Datei als Attribute des Portlet-Elements (`<bw:portlet>`). Es handelt sich um die Attribute `width` und `height`. Wir können beiden den Wert „*“ (Sternchen bzw. Asterisk) zuweisen oder beide Attribute löschen. Nach Speichern der Änderungen genügt ein Neuladen („reload“) des Portlets mit Hilfe des entsprechenden Symbols innerhalb der Portlet-Titlebar (siehe auch Abb. 11), um die Änderungen innerhalb des Prozesses darzustellen.

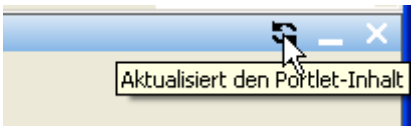



**Abb. 11: Reload-Button innerhalb der Portlet-Titlebar**

Damit ist unser Beispiel-Dialog abgeschlossen. Die zugehörige Datei ist nachfolgend vollständig dargestellt.

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:portlet
   xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml"
   id="Test_SampleDialog"
   title="Partner suchen"
   target="cont">

  <bw:label id="BWLabel1">
    <bw:content>
      <bw:t>Partner suchen</bw:t>
    </bw:content>
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
            eox="13" eoy="13" ecx="230" ecy="36"/>
  </bw:label>
  <bw:label id="BWLabel2">
    <bw:content>
      <bw:t>Bitte geben Sie die gewünschte Suchkriterien an und drücken
       anschließend den "Suchen"-Button. Bei allen Suchkriterien wird
       automatisch ein "*"-Wildcard angehängt. Einzelne Buchstaben
       können zusätzlich durch das "?"-Wildcard repräsentiert werden.
       Auch das "*"-Wildcard kann mitten im Text als Platzhalter für
       beliebige und beliebig viele Zeichen verwendet werden.
     </bw:t>
    </bw:content>
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
            eox="36" eoy="36" ecx="605" ecy="96"/>
  </bw:label>
  <bw:label id="BWLabel3">
    <bw:content>
      <bw:t>Nachname</bw:t>
    </bw:content>
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
            eox="70" eoy="106" ecx="170" ecy="129"/>
  </bw:label>
  <bw:label id="BWLabel4">
    <bw:content>
      <bw:t>Vorname</bw:t>
    </bw:content>
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
            eox="70" eoy="129" ecx="170" ecy="152"/>
  </bw:label>
  <bw:label id="BWLabel5">
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:content>
    <bw:t>PLZ</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="70" eoy="152" ecx="170" ecy="175"/>
</bw:label>
<bw:label id="BWLabel6">
  <bw:content>
    <bw:t>Ort</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="70" eoy="175" ecx="170" ecy="198"/>
</bw:label>
<bw:label id="BWLabel7">
  <bw:content>
    <bw:t>Art der Suche</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="70" eoy="276" ecx="170" ecy="299"/>
</bw:label>
<bw:entry id="BWEntry1" tabindex="8">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="108" ecx="415" ecy="128"/>
</bw:entry>
<bw:entry id="BWEntry2" tabindex="9">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="131" ecx="415" ecy="151"/>
</bw:entry>
<bw:entry id="BWEntry3" tabindex="10">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="154" ecx="248" ecy="174"/>
</bw:entry>
<bw:entry id="BWEntry4" tabindex="11">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="177" ecx="415" ecy="197"/>
</bw:entry>
<bw:choice id="BWChoice1" tabindex="12" edit="false">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="278" ecx="415" ecy="299"/>
</bw:choice>
<bw:button id="BWButton1" tabindex="13">
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:content>
    <bw:t>Suchen</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="554" eoy="420" ecx="629" ecy="443"/>
</bw:button>
<bw:button id="BWButton2" tabindex="14">
  <bw:content>
    <bw:t>Abbrechen</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="473" eoy="420" ecx="548" ecy="443"/>
</bw:button>
<bw:checkbox id="BWCheckbox1" tabindex="15">
  <bw:content>
    <bw:t>Erweiterte Suche nach Umlauten verwenden</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="203" ecx="533" ecy="227"/>
</bw:checkbox>
<bw:checkbox id="BWCheckbox2" tabindex="16">
  <bw:content>
    <bw:t>Auch nach "ehemaligen Namen" suchen</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="224" ecx="533" ecy="248"/>
</bw:checkbox>
<bw:label id="BWLabel8">
  <bw:content>
    <bw:t>Suchen nach</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="70" eoy="299" ecx="170" ecy="322"/>
</bw:label>
<bw:checkbox id="BWCheckbox3" tabindex="18">
  <bw:content>
    <bw:t>Interessent / Kunde</bw:t>
  </bw:content>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="176" eoy="299" ecx="415" ecy="323"/>
</bw:checkbox>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:checkbox id="BWCheckbox4" tabindex="19">
    <bw:content>
      <bw:t>Mandant / Produktgeber</bw:t>
    </bw:content>
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
            eox="176" eoy="320" ecx="415" ecy="344"/>
  </bw:checkbox>
  <bw:checkbox id="BWCheckbox5" tabindex="20">
    <bw:content>
      <bw:t>Versicherte Person</bw:t>
    </bw:content>
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
            eox="176" eoy="341" ecx="415" ecy="365"/>
  </bw:checkbox>
</bw:portlet>
```

Bis zur Produktionsreife wären noch folgende (wenige kleine) Schritte erforderlich bzw. sinnvoll:

- Semantisch sinnvolle Namen (IDs) für jedes Element (sinnvoll); für die Namensgebung gibt es innerhalb von *B-Wise* entsprechende Konventionen
- Definition der Zuordnung zwischen Control (z.B. Eingabefeld) und Datenobjekt (erforderlich); diese erfolgt mit Hilfe des „`datasrc`"-Attributs, das einem Control zugewiesen wird
- Definition von Styles z.B. für die Überschrift (sinnvoll)

## 2.3 Anzeigeseite

Auch die Anzeigeseite ist ein Portlet, das jedoch nur ein einziges Element, nämlich das `<bw:browser>`-Element enthält. Darin befindet sich eine XHTML-Seite, die mit BML-Tags kombiniert ist.

Das Portlet selbst wird am besten mit dem *B-Wise* IDE Designer erstellt. Wählen Sie hierzu aus der Palette der verfügbaren Elemenet das Browser-Element und plazieren es so innerhalb des Portlets, dass es die komplette nutzbare Fläche (Clientarea) des Portlets ausfüllt. Am besten definieren sie auch das Scaling so, dass die rechte untere Ecke am rechten unteren Rand des Portlets verankert wird; damit wird der Browser zusammen mit dem Portlet skaliert (vergrößert oder verkleinert). Weitere Elemente sind nicht erforderlich. Der Code des fertigen Portlets sieht wie folgt aus (der Code ist letztlich so einfach, dass er auch direkt mit einem gewöhnlichen Editor erstellt werden kann):

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:portlet
   xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml"
   id="Test_Browser"
   title="Browser"
   target="cont">

  <bw:browser id="BR_1" tabindex="1">
    <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
             eox="0" eoy="0" ecx="0" ecy="0"/>
        <bw:content>
            <bw:include src="Test_SampleShow.bml" />
        </bw:content>
    </bw:browser>

</bw:portlet>
```

Der fett gedruckte Dateiname ist der Name der noch zu erstellenden Anzeigedatei (XHTML-Seite). Speichern Sie den Code in einer Datei namens „Test_SampleShow.plt“.

Die XHTML-Seite kann mit jedem ASCII-Editor (z.B. mit dem *B-Wise* IDE Editor) erstellt werden. Auch viele, spezialisierte HTML- oder XML-Editoren eignen sich zur Erstellung von Anzeigeseiten, da sie meist HTML-fremde Tags einfach ignorieren. Dies erlaubt die interaktive, grafische Erstellung der Anzeigeseiten mit Tools wie z.B. MS Frontpage, Adobe GoLive oder NetObjects Fusion sowie das nachträgliche Erweitern um spezielle BML-Tags. Der *B-Wise* IDE Editor bietet den Vorteil integrierter Tests, d.h. die Möglichkeit einer integrierten Transformation sowie des Previews der transformierten Seite.

Um die Erstellung und das Testen der Anzeigeseiten zu vereinfachen, bietet B-Wise die Möglichkeit, innerhalb der Skriptsprache Groovy auch einfache Services zu definieren. Zusätzlich können die Daten einer BML-Dialog- oder -Anzeigeseite nicht nur aus dem Objektmodell sondern ebenfalls aus einer XML-Datei gelesen werden. Die XML-Datei dient dann als Schnittstelle zwischen Dialog- oder Dokument-Designer und Model-Entwickler.

Das folgende Beispiel geht von einem einfachen Datenmodell aus, das in der folgenden XML-Datei dargestellt ist, die das (Beispiel-) *B-Wise*-Objektmodell repräsentieren soll:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Test>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<Kunde>
  <Name>Mustermann</Name>
  <Vorname>Max</Vorname>
  <GebDat>22.09.1953</GebDat>
  <AdresseListe anzahl="2">
    <Adresse typ="1">
      <Strasse>Hauptstraße 2</Strasse>
      <PLZ>26389</PLZ>
      <Ort>Wilhelmshaven</Ort>
    </Adresse>
    <Adresse typ="2">
      <Strasse>Promenade 1</Strasse>
      <PLZ>20233</PLZ>
      <Ort>Hamburg</Ort>
    </Adresse>
  </AdresseListe>
</Kunde>

</Test>
```

Speichern Sie diese Datei unter dem Namen „Test_SampleData.xml“ ab.

Jetzt können wir die eigentliche Anzeigeseite erstellen. Die folgende XHTML-Seite könnte zur Anzeige der Daten dienen (die BML-Tags sind zur besseren Visualisierung fett gedruckt):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "dtd\xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
      xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

<bw:usecls name="com.biss.service.common.FileReader" />
<bw:var name="Data" expr="readFile( 'Test_SampleData.xml')" />

<head>
  <title>Beispieldatei Kundenanzeige</title>
</head>

<body>
  <h1>Kundendaten</h1>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <p>Die folgende Seite enthält einige Daten zum Kunden.</p>
  <table>
    <colgroup>
      <col width="40%" />
      <col width="60%" />
    </colgroup>
    <tr>
      <td>Name</td>
      <td><bw:var expr="Kunde.Name" /></td>
    </tr>
    <tr>
      <td>Vorname</td>
      <td><bw:var expr="Kunde.Vorname" /></td>
    </tr>
    <tr>
      <td>Geburtsdatum</td>
      <td><bw:var expr="Kunde.GebDat" /></td>
    </tr>
    <bw:foreach element="Kunde.AdresseListe.*Adresse">
      <tr>
        <td>Strasse_<bw:var expr="pos()" /></td>
        <td><bw:var expr=".Strasse" /></td>
      </tr>
      <tr>
        <td>Postleitzahl_<bw:var expr="pos()" /></td>
        <td><bw:var expr=".PLZ" /></td>
      </tr>
      <tr>
        <td>Ort_<bw:var expr="pos()" /></td>
        <td><bw:var expr=".Ort" /></td>
      </tr>
    </bw:foreach>
  </table>
</body>
</html>
```

Erstellen Sie diese Datei mit Hilfe eines Texteditors und speichern sie unter dem Namen „Test_SampleShow.bml“ ab.

Wie Sie sicherlich schnell erkennen werden, handelt es sich bei der Anzeigeseite um eine gewöhnliche XHTML-Datei mit zusätzlichen Tags bzw. Elementen, die sich alle im "bw"-


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Namensraum befinden (z.B. "<bw:var ...>"). Die Betonung liegt hier auf dem "X" in XHTML, da BML-Anzeigeseiten mit Hilfe eines XML-Parsers geparst werden und daher der XHTML-Konvention genügen müssen (siehe hierzu entsprechende Informationen sowie Dokumentation auf http://www.w3.org).

Mit wenigen Ausnahmen (die alle in den entsprechenden Abschnitten beschrieben sind) können BML-Tags an beliebigen Stellen im XHTML-Code vorkommen.

Wenden wir uns kurz den BML-Tags im obigen Beispiel zu. Die Templatedatei verwendet zwei BML-Tags:

- <bw:var> und
- <bw:foreach>

<bw:var> ermöglicht u.a. den Zugriff auf das *B-Wise*-Objektmodell bzw. die entsprechende XML-Datei. Nach der Transformation wird dabei der Inhalt des im Attribut "expr" aufgeführten Ausdrucks an die entsprechende Stelle der XHTML-Datei eingefügt (das "<bw:var>"-Konstrukt wird nicht in die HTML-Ausgabe kopiert). Im obigen Beispiel wird also die erste Zeile der Tabelle in der zweiten Spalte den Namen des Kunden, hier "Mustermann" enthalten. Es ist wichtig anzumerken, dass der Transformer das Root-Element der XML-Datei ignoriert ! Die einzelnen Elemente werden innerhalb der Pfadangabe mit Hilfe eines Punktes ('.') voneinander getrennt (entspricht dem Schrägstrich innerhalb von XSL).

<bw:foreach> stellt ein Schleifenkonstrukt dar. Hierzu benötigt es innerhalb des *B-Wise*-Objektmodells (bzw. innerhalb der entsprechenden XML-Datei) eine Liste, die es abarbeitet. Dabei wird der innerhalb des <bw:foreach>...</bw:foreach>-Konstrukts stehende Inhalt für jedes Element der Liste wiederholt. Innerhalb der <bw:foreach>-Anweisung muss der entsprechende Pfadausdruck zum gewünschten Listenelement angegeben werden.

Um nun zur Darstellung zu gelangen muss die BML-Templatedatei mit Hilfe des Objektmodells bzw. (wie in diesem Beispiel) mit Hilfe der XML-Datei transformiert werden. Dies erfolgt mit Hilfe des *B-Wise*-XML-Transformers. Durch die Anweisungen

```
<bw:usecls ... />
```

und

```
<bw:var name="data" expr="readFile('Test_SampleData.XML') " />
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

innerhalb der XHTML-Datei verwendet der Transformer die darin angegebene Datei statt des *B-Wise*-Objektmodells.

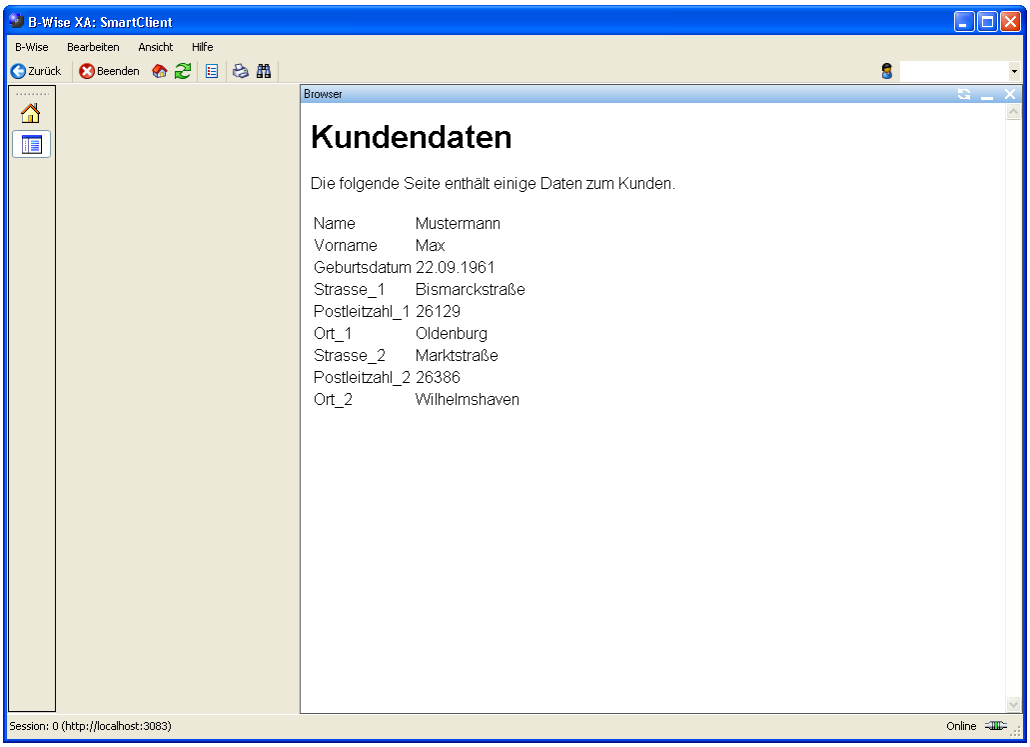



**Abb. 12: Beispieldatei im SmartClient**

Die Anzeige ist noch nicht besonders anspruchsvoll, so dass ein wenig „Styling“ angebracht wäre. Dies erfolgt mit Hilfe entsprechender CSS-Dateien und entspricht dem Vorgehen bei „klassischen“ XHTML-Seiten.

Alternativ zur Speicherung der Daten innerhalb einer XML-Datei und Zugriff auf die Datei mit Hilfe des `<bw:usecls>`-Elements sowie des darauffolgenden `<bw:var>`-Elements können wir die Daten auch innerhalb des Script-Moduls abbilden. Dies ist in der folgenden Code-Sequenz dargestellt:

```
import com.biss.dlgsys.ifc.PortletDescrs;
import com.biss.process.GUIProcess;

class TestProzess
   extends GUIProcess {

   public TestKunde Kunde = new TestKunde();
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
   public TestProzess() {
      super();
   }

   public String getName() {
      return "Einfacher Testprozess";
   }

   public String getPreferredPortal() {
      return "System_Standard.por";
   }

   public PortletDescrs definePortlets() {
      PortletDescrs portlets = super.definePortlets();
      portlets.add( "Test_SampleShow.plt");
      return portlets;
   }

   public void initialize() {
      super.initialize();
   }

   public void start() {
      super.start();
   }
}

class TestKunde {
   public String  Nachname;
   public String  Vorname;
   public String  GebDat;
   public Adresse[] AdressListe;

   public TestKunde () {
      Nachname = "Mustermann";
      Vorname  = "Max";
      GebDat   = "22.09.1961";
      AdressListe = new TestAdresse[2];
         AdressListe[0].PLZ = "26129";
         AdressListe[0].Ort = "Oldenburg";
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
        AdressListe[0].Strasse = "Bismarckstraße";
        AdressListe[0].HausNr = "22";
        AdressListe[1].PLZ = "26386";
        AdressListe[1].Ort = "Wilhelmshaven";
        AdressListe[1].Strasse = "Marktstraße";
        AdressListe[1].HausNr = "204";
   }
}

class TestAdresse {
   public int     PLZ;
   public String  Ort;
   public String  Strasse;
   public String  HausNr;
}
```

Es handelt sich um den bereits bekannten Test-Prozess, den wir im ersten Beispiel (Suchdialog) verwendet und für unsere neuen Testzwecke entsprechend modifiziert haben. Die Änderungen gegenüber der ersten Version sind fett hevorgehoben.

Im Wesentlichen handelt es sich um die Verwendung des neuen Portlets „Test_SampleShow.plt“ sowie das Hinzufügen der Datenklassen TestKunde und TestAdresse zur Anzeige der entsprechenden Daten innerhalb der XHTML-Seite.

In den folgenden Abschnitten wird ausführlich auf die Elemente bzw. Sprachkostrukte der BML eingegangen.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

deafult

Kapitel 3

# BML-Sprachkonstrukte

## 3.1 Allgemeine Hinweise

Alle Tags (sowie deren Attribute) werden klein geschrieben. Die BML-Tags befinden sich innerhalb des Namensraums `bw`.

> **Hinweis:** Da es sich beim BML um XML handelt, sind selbstverständlich alle innerhalb von XML erlaubten Konstrukte ebenfalls in BML möglich. Hierzu gehören insbesondere Kommentare (`<!-- Dies ist ein Kommentar -->`) sowie Entities (z.B. ` ` entspricht einem unbedingten Lerzeichen).

BML kennt - in alphabetischer Reihenfolge aufgezählt - in der Version 4.03 folgende Sprachkonstrukte (Elemente):

| Tags | Bedeutung |
|---|---|
| `<bw:cjscript>` `</bw:cjscript>` | Erlaubt die Definitin von JScript-Funktionen innerhalb einer BML-Seite, die anschließend vom Client ausgeführt werden |
| `<bw:element>` `</bw:element>` | Dient der selektiven Behandlung abgeleiteter Dialogseiten, zusammen mit dem <bw:page> oder auch <bw:include> Konstrukt. |
| `<bw:foreach>` `</bw:foreach>` | Schleifenkonstrukt; ermöglicht die Iteration über eine Liste von Werten bzw. Objekten |
| `<bw:group>` `</bw:group>` | Hilfskonstrukt beim Inkludieren von BML-Dateien (siehe auch <bw:include />) |
| `<bw:if>` `</bw:if>` | Bedingung; ermöglicht die bedingte Transformation von Code-Segmenten |
| `<bw:ig>` `</bw:ig>` | Ignore-Section; ermöglicht die Definition von XHTML-Code-Bereichen, die bei einer Transformation ignoriert werden (insbesondere für die Erstellung von Prototypen geeignet) |
| `<bw:include />` | Ermöglicht das Inkludieren einer wohlgeformten BML-Datei an einer beliebigen Stelle innerhalb des XMl_DD-Codes; dient darüberhinaus der Realisierung von Mehrfach-Vererbung („multiple |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | inheritance“) |
| `<bw:lookup />` | Hilfskonstrukt zur Lookup-Table (siehe unten) |
| `<bw:lutab>`<br>`</bw:lutab>` | Lookup-Table; ermöglicht die Definition von Lookup-Tabellen zur einfachen Konvertierung bzw. Zuweisung von Werten |
| `<bw:page>`<br>`</bw:page>` | Dient der Definition einer abgeleiteten Dialogseite (zusammen mit dem Konstrukt <bw:element>; siehe auch dieses) |
| `<bw:script>`<br>`</bw:script>` | Ermöglicht die Definition von JavaScript-Funktionen, die während der Transformation verfügbar sind |
| `<bw:switch>`<br>`</bw:switch>` | Ermöglicht die Konstruktion einer Fallunterscheidung inkl. eines Sammelstatements (<bw:default>). |
| `<bw:te />` | Hilfskonstruktur zur Lookup-Table |
| `<bw:usecls />` | Gibt an, welche Java-Klasse für den Aufruf externer Funktionen verwendet werden soll |
| `<bw:var>`<br>`</bw:var>` | Variablen; ermöglicht den Zugriff auf das *B-Wise* Objektmodell (DOM) sowie die Definition von Variablen |

## 3.2 <bw:cjscript>

Die Syntax des `<bw:cjscript>`-Elements ist wie folgt:

```
<bw:cjscript copts="fast" name="name">
  <![CDATA[
  …
  ]]>
</bw:cjscript>
```

Das Element erlaubt die Definition beliebiger CJScript- oder C#-Funktionen innerhalb einer BML-Seite, die anschließend vom Client ausgeführt werden. Damit gibt es einen Unterschied zum <bw:script>-Element, dessen Javascript-Code zum Zeitpunkt der Transformation (!) ausgeführt wird.

Clientseitige Skripte erlauben eine Reihe nützlicher Funktionen wie z.B. die Eingabefilterung oder die „Dynamifizierung“ des GUI, wie z.B. das Auf- oder Zuklappen definierter Dialogbereiche. Das folgende Beispiel möge das verdeutlichen:


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:cjscript>
    <![CDATA[
    function refresh() {
     var firm, pers, vis, sel;
     sel = Doc.getElementById( "CH_Firmenkunde");
     firm= Doc.getElementById( "CP_Firma");
     pers= Doc.getElementById( "CP_BasisNatPers");
     vis = (sel.getProperty( "SelectedIndex") == 0);
     if ( firm.getProperty( "Visible") != vis ) {
       firm.setProperty( "Visible", vis);
       pers.setProperty( "Visible", !vis);
       firm.Parent.layout();
     }
     else if ( pers.getProperty( "Visible") == vis ) {
       firm.setProperty( "Visible", !vis);
       pers.setProperty( "Visible", vis);
       firm.Parent.layout();
     }

   }
      ]]>
</bw:cjscript>
```

Dem Entwickler steht eine Reihe von Methoden zur Verfügung, um mit dem clientseitigen DOM zu interagieren. Die folgende Aufzählung gibt einen kurzen Überblick:

Auf der element-Ebene:

```
void setProperty( string prop, string val);
object getProperty( string prop);
void addEventHandler( string name, string method);
object triggerEvent ( string name);
void setStyleRef ( string ref);
void processCommand( string cmd);
void reflowLayout();
void layout();
void showBubbleHelp( int time);
void validate();
void refresh();
void setFocus();
int getPreferredWidth();
int getPreferredHeight();
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Auf der document: ( Doc. ) - Ebene

```
object getElementById( string id);
void debug( string msg);
void doLayout();
void processCommand( string cmd);
```

Global:

```
void alert( string msg, string title);
```

## 3.3 <bw:element>

`<bw:element>` ist ein Konstrukt zur Realisierung selektiver Modifikationen bei Ableitungen. Eine nähere Beschreibung mit Beispielen ist im Abschnitt 5 (*BML-Inheritance*) enthalten.

### Syntax

Die Syntax des `<bw:element>`-Tags ist wie folgt:

```
<bw:element mode="replace" id="id">
  … XHTML-Code…
</bw:element>
```

Die Attribute haben folgende Bedeutung:

| Attribut | Bedeutung |
|---|---|
| `mode` | Ermöglicht die Definition des Verarbeitungsmodus für das betreffende `<bw:element>`; aktuell stehen folgende Werte zur Verfügung:<br>`replace` Der Inhalte des `<bw:element>` Tags ersetzt den Inhalt des mit `id` bezeichneten oder mittels `path` adressierten Elements der Basisseite; ein `replace` mit einem leeren `<bw:element>` entspricht einem delete (Löschen)<br>`insert` Der Inhalt des `<bw:element>`-Tags wird als Subtree an den Inhalt des mit `id` bezeichneten oder mittels `path` adressierten Elements angehängt |
| `id` | Ermöglicht die Adressierung eines definierten Elements der Basisseite; dieses Element muss über ein entsprechendes ID-Attribut verfü- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | gen; `id` ist alternativ zu `path` zu verwenden ! |
| `path` | Ermöglicht die Adressierung eines definierten Elements der Basisseite mit Hilfe eines Pfadausdrucks; `path` ist alternativ zu `id` zu verwenden ! |

**Beispiel**

```
<?xml version="1.0" encoding="iso-8859-1"?>

<bw:page extends="BWBase.BML">
  <!-- Page-spezifische Styles -->
  <bw:element mode="insert" path="html.head">
    <style>
      @import url( "styles/bwportal.css");
    </style>
  </bw:element>

  <!-- Navigation -->
  <bw:element mode="replace" id="#navigation">
    <div id="navigation">
      <p class=header"><bw:var expr="$BWNP_I_Title" /></p>
      <ul>
      <bw:foreach element="SysCmds.*Command">
        <li>
          <a href="{cmd( .Name)}" class="menue">
            <bw:var expr=".Label" />
          </a>
        </li>
      </bw:foreach>
      </ul>
    </div>
  </bw:element>
</bw:page>
```

# 3.4 <bw:foreach>

In vielen Fällen müssen mehrere Objekte einer Liste z.B. innerhalb von HTML-Tabellen dargestellt werden. Hierfür eignet sich der `<bw:foreach>`-Tag. Er iteriert über die innerhalb der Attribute angegebene Liste und überträgt den innerhalb des `<bw:foreach>`

© Copyright BISS GmbH, 2001-2012 All Rights reserved

`... </bw:foreach>`-Blocks liegenden XHTML-Code für jedes Element der Liste genau einmal in die Ausgabe.

### Syntax

Die Syntax des `<bw:foreach>`-Tags ist wie folgt:

```
<bw:foreach element="Pfadausdruck" maxn="Count" autodiv="Name">
     ... XTML-Code...
</bw:foreach>
```

Die Attribute haben folgende Bedeutung:

| Attribut | Bedeutung |
|---|---|
| `element` | Gibt einen vollständigen Ausdruck an (siehe auch Abschnitt 4.5), der eine Liste sowie das entsprechende Listen-Element enthalten muss, also z.B. für das Beispiel des Abschnitts 2: `Partner.AdresseListe.*Adresse` Eine Besonderheit stellt der Ausdruck „[0..3]“ dar, also zwei Zahlenwert in eckigen Klammer getrennt durch zwei Punkte. Sie definierten einen konstanten Schleifenzählen (im Beispiel von 0 bis 3 inklusive, also 4 Elemente), was insbesondere bei Prototypen nützlich sein kann. |
| `maxn` | (Optional) Gibt die maximale Anzahl der auszugebenden Elemente an; wird dieses Attribut verwendet, werden maximal `"Count"` Elemente der Liste in die Ausgabe übernommen; wird das Attribut weggelassen, werden alle Elemente der Liste in die Ausgabe übernommen. |
| `autodiv` | (Optional) Falls vorhanden, wird automatisch eine separate `<bw:div>`-Sektion generiert, welche mit Hilfe der *B-Wise*-JavaScript-Funktion `toggleList` aufgeblättert werden kann (siehe auch Beispiel in Anhang D). Die Verwendung von `"autodiv"` macht nur bei einem ebenfalls angegebenen `"maxn"` Sinn. |

Innerhalb des Schleifen-Blocks kann auf die einzelnen (Sub-)Elemente des Listenelements mit Hilfe einer abgekürzten Schreibweise zugegriffen werden. Die explizite Schreibweise, welche den kompletten Pfadausdruck verwendet (d.h. im Beispiel des Abschnitts 2 `"Kunde.AdresseListe.Adresse.PLZ"`) kann in der aktuellen BML-Version nicht verwendet werden ! Die abgekürzte Schreibweise impliziert den Pfad zu den einzelnen (Sub-) Elementen des Listen-Elements, so dass diese einfach mit einem vorangestellten Punkt angegeben werden können (z.B. `".PLZ"`).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Der <bw:foreach>-Tag stellt zusätzlich die interne Funktion pos() zur Verfügung, welche die aktuelle Position (0-basiert) innerhalb der Liste zurückgibt. Das erste Listenelement hat dabei die Position 0, das zweite die Position 1, usw. Die Funktion kann ausschließlich innerhalb eines <bw:foreach>-Blocks verwendet werden und bezieht sich immer auf die unmittelbar darüber liegende <bw:foreach>-Anweisung (wichtig bei verschachtelten Schleifen !).

**Beispiel**

```
<bw:foreach element="Kunde.AdresseListe.*Adresse">
  <tr>
    <td>Strasse_<bw:var expr="pos()" /></td>
    <td><bw:var expr=".Strasse" /></td>
  </tr>
  <tr>
    <td>Postleitzahl_<bw:var expr="pos()" /></td>
    <td><bw:var expr=".PLZ" /></td>
  </tr>
  <tr>
    <td>Ort_<bw:var expr="pos()" /></td>
    <td><bw:var expr=".Ort" /></td>
  </tr>
</bw:foreach>
```

Die Schleife im obigen Beispiel iteriert über die Liste "Kunde.AdresseListe", welche Elemente mit Namen "Adresse" enthalten muss. Innerhalb der Schleife wird mit Hilfe der abgekürzten Schreibweise auf die einzelnen (Sub-)Elemente des Elements "Adresse" zugegriffen.

## 3.5 <bw:group>

Das <bw:group>-Tag dient der Klammerung von BML- bzw. XHTML-Statements innerhalb einer XHTML- bzw. BML-Datei, die in eine andere BML-Datei inkludiert werden sollen (siehe auch das <bw:include>-Tag).

**Syntax**

```
<bw:group>
   ... XHTML- bzw. BML-Code ...
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
</bw:group>
```

Das <bw:group>-Tag wird normalerweise ausschließlich in Dateien verwendet, die in eine andere BML-Datei inkludiert werden sollen. Es wird benötigt, da die inkludierte Datei selbst eine wohlgeformte XML- (bzw. XHTML-) Datei sein muss und daher als solche lediglich ein Root-Element besitzen darf.

**Beispiel**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:!-- DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "dtd\xhtml1-strict.dtd" -->

<bw:group xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
      xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">
  <tr>
    <td class="head2"> </td>
    <td class="head2">Feldname</td>
    <td class="head2">Basis</td>
    <td class="head2"> </td>
    <td class="head2">Dublette</td>
  </tr>
  <tr>
    <td class="head2"> </td>
    <td class="head2"> </td>
    <td class="head2">bleibt erhalten</td>
    <td class="head2"> </td>
    <td class="head2">wird gelöscht</td>
  </tr>
</bw:group>
```

## 3.6 <bw:if>

Das <bw:if>-Tag dient der konditionalen Transformation von XHTML-Code. Der innerhalb des <bw:if> ... </bw:if>-Blocks liegende XHTML-Code wird nur dann in die Ausgabe übernommen, wenn die IF-Bedingung wahr ist.

**Hinweis:** Innerhalb der BML existiert kein "else"-Konstrukt.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Hinweis:** Häufig kann anstatt mehrerer IF-Konstrukte eine Lookup-Tabelle verwendet werden (siehe daher Abschnitte 3.9 und 4.7).

### Syntax

Die Syntax des `<bw:if>`-Tags ist wie folgt:

```
<bw:if expr="Ausdruck">
    ... XHTML-Code ...
</bw:if>
```

`<bw:if>` hat ein einziges Attribute, (`expr`), das die Bedingung der IF-Anweisung enthält. In der Bedingung können alle in BML gültigen Ausdrücke (siehe Abschnitt 4) stehen, z.B. Funktionsaufrufe, logische Operatoren oder Stringvergleiche. Bei boolschen Variablen kann zur Abkürzung der Schreibweise auf den expliziten Vergleich mit der boolschen Konstanten ('true' oder 'false') verzichtet werden (siehe auch Beispiel unten).

| Attribut | Bedeutung |
|---|---|
| `expr` | Definiert den Ausdruck der IF-Bedingung; der Ausdruck muss ein boolscher Ausdruck sein und die Werte wahr (`true`) oder falsch (`false`) liefern (siehe auch BML-Ausdrücke in Abschnitt 4) |

### Beispiel

```
<bw:if expr="Partner.Alter > 2">
  <table>
    <tr><bw:td>Zeile 1</td></tr>
    <tr><bw:td>Zeile 2</td></tr>
  </table>
</bw:if>

<bw:!-- Im Folgenden die abgekürzte Schreibweise bei boolschen Variablen
      bzw. bei numerischen oder Stringvariablen
-->
<bw:var name="flag"   expr="true" />
<bw:var name="alter"  expr="0"    />
<bw:var name="string" expr="Test" />

<bw:if expr="$flag">
  <p>Dieser Absatz wird ausgegeben</p>
</bw:if>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:if expr="$alter">
  <p>Dieser Absatz wird nicht ausgegeben</p>
</bw:if>

<bw:if expr="$string">
  <p>Dieser Absatz wird ausgegeben</p>
</bw:if>
```

Im obigen Beispiel wird die Tabelle nur dann transformiert (in die Ausgabe übernommen), wenn die Variable `Partner.Alter` einen Wert größer 2 hat. Im zweiten Beispiel kann auf den expliziten Vergleich der Variablen "`flag`" auf den Wert `'true'` bzw. den Variablen "`alter`" und "`string`" auf die Werte "`0`" bzw. "`Test`" verzichtet werden, sofern lediglich eine Unterscheidung zwischen 0 und Nicht-0 bzw. Leerstring und Nicht-Leerstring erforderlich ist.

## 3.7 <bw:ig>

Das <bw:ig>-Tag dient der Definition sog. "Ignore"-Blöcke, die bei der Transformation nicht in die Ausgabe übernommen werden. Ignore-Blöcke sind sinnvoll, wenn innerhalb der XML-Templatedatei Informationen visuell (z.B. im Rahmen eines Prototypings) angezeigt jedoch im späteren (Produktiv-) Einsatz nicht benötigt werden.

### Syntax

Die Syntax des `<bw:ig>`-Tags ist wie folgt:

```
<bw:ig>
  <!-- XHTML-Code, der nicht in die Ausgabe übernommen wird -->
</bw:ig>
```

Das `<bw:ig>`-Tag hat keine Attribute.

Beispiel

```
<bw:ig>
      Dieses Portal dient der Anzeige von Daten zu einem Partner;
      es enthält sehr viele Informationen auf engem Raum und gibt
      dadurch eine gute Übersicht über den Partner-<bw:i>Status</bw:i>.
      Mit Hilfe von Hyperlinks gelangt man zu weiteren Seiten sowie
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      zu entsprechenden Bearbeitungsfunktionen.
</bw:ig>
```

Der im Beispiel zwischen `<bw:ig>` und `</bw:ig>` stehende Text wird nicht in die Ausgabe übernommen.

# 3.8 <bw:include>

Das `<bw:include>`-Tag inkludiert eine wohlgeformte BML- bzw. XHTML-Datei an eine beliebige Stelle innerhalb eines anderen BML-Dokuments.

Zusätzlich stellt `<bw:include>` die Basis für die „Vererbung“ innerhalb von BML-Dateien bzw. den assoziierten DOM-Strukturen dar. Nehmen wir an, eine existierende BML/HTML-Datei soll wieder verwendet, dabei jedoch an bestimmten Stellen modifiziert werden. Dabei können Elemente (oder ganze Subtrees des DOM) hinzugefügt, gelöscht oder geändert bzw. Element-Attribute hinzugefügt oder gelöscht werden.

Die Ersetzung basiert auf einem erweiterten `<bw:include>`. Dabei wird der Zugriff auf das DOM der inkludierten BML-Datei ermöglicht. Ein Element (oder auch ein Subtree von Elementen) kann dann über einen BML-Pfad-Ausdruck oder das `id`-Attribut des Elements bzw. Subtrees adressiert werden.

Innerhalb der inkludierenden Datei wird ein BML-Element `<bw:element>` verwendet, welches ein (oder mehrere) BML/HTML-Elemente adressiert und je nach Wert des Attributs 'mode' (Werte: "replace" und "insert") die verschiedenen Modifikations-Operationen erlaubt. Zusätzlich existiert ein weiteres BML-Element `<bw:attr>`, das es erlaubt, Attribute zu ändern bzw. zu löschen.

Die verschiedenen `<bw:element>`-Anweisungen werden dabei sequentiell abgearbeitet; sie werden jeweils auf das Resultat der vorhergehenden Ersetzung angewandt.

**Syntax**

```
<bw:include src="dateiname" />
```

Bei der inkludierten Datei muss es sich um eine wohlgeformte BML- bzw. XHTML-Datei handeln (siehe auch das `<bw:group>`-Tag). Der Inhalt der im Attribut `'src'` angegebenen Datei wird lexikalisch an die Stelle des `<bw:include>`-Tags eingefügt.

```
<bw:include src="dateiname">
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:element id="mydiv" mode="mymode">
     ...
  </bw:element>
  <bw:element path="mypath" mode="mymode">
           <bw:attr name="attrname" value="newval" />
  </bw:element>
</bw:include>
```

Für die Identifikation eines Elements (innerhalb von `<bw:element>`) kann sowohl die ID (Wert des `id`-Attributs) des Elements als auch der vollständige Pfad zum Element verwendet werden (siehe auch Beispiele weiter unten). In der BML-Version 4.0.3 sind zwei Modi für das Attribut `mode` vorgesehen:

| Attribut | Bedeutung |
|---|---|
| append | Fügt die neuen Anteile an das Element an |
| replace | Ersetzt den Wert des aktuellen Elements durch die neuen Anteile; wird als neue Anteile gar nichts angegeben und wird das mode-Attribut mit dem Wert „replace“ verwendet, entspricht dieser Modus einem „delete“. |

Wird das mode-Attribut weggelassen, wird das Element bzw. der entsprechende Subtree unverändert in das inkludierende Dokument übernommen und ersetzt ggf. ein existierendes Element mit derselben ID bzw. demselben Pfad. Dieser Modus kann auch für das Ändern der Attribute des entsprechenden Elements verwendet werden. Dabei wird das BML-Tag `<bw:attr>` verwendet, welches den Namen des Attributs sowie den neuen Wert als Ausdruck (expr) erhält.

### Beispiel

```
<bw:include src="MyHeader.BML" />
```

Weitere Beispiele für die Verwendung von `<bw:include>` zusammen mit `<bw:element>` und `<bw:attr>`.

```
<bw:include src="mydoc">
<!-- Ersetzen eines Elements (evtl. durch mehrere) -->
<bw:element mode="replace" path="html.body.table.tbody.*td[pos()=0]">
  <td> ... </td>
  <td> ... </td>
</bw:element>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<!-- Löschen von Elementen = Ersetzen durch nichts -->
<bw:element mode="replace" path="html.body.table.tbody.*td" />

<bw:!-- Hinzufügen von Elementen -->
<bw:!-- Optionale Angabe der Einfügeposition bei Listen durch pos=".." -
->
<bw:element mode="insert" path="html.body.table.tbody" pos="0">
  <td> ... </td>
  <td> ... </td>
  <td> ... </td>
</bw:element>

<!-- Ändern von Attributen -->
<!-- Ersetzen /Hinzufügen eines Attributs -->
<!-- Löschen eines Attributs, entspricht value="" -->
<bw:element mode="changeAttrs" path="html.body.table.tbody.td">
  <bw:attr name="colspan" value="3" />
  <bw:attr name="align" />
</bw:element>
...
</bw:include>
```

<bw:include> ermöglicht außerdem die Übergabe von Argumenten an die inkludierte Datei. Dabei gibt es zwei grundsätzliche Vorgehensweise:

```
<bw:var name=“arg1“ expr=“22“ />
<bw:var name=“arg2“ expr=“‘Konstanter String‘“ />
<bw:include src=“Datei.inc“ />
```

und

```
<bw:include src=“Datei.inc“>
  <bw:var name=“arg1“ expr=“22“ />
  <bw:var name=“arg2“ expr=“‘Konstanter String‘“ />
</bw:include>
```

Der Unterschied zwischen den aufgezeigten Vorgehensweisen liegt in der Sichtbarkeit (demo „Scope“) der beiden Argumente „arg1“ und „arg2“. Während im ersten Fall diese Argumente einen globalen Scope haben, also ab ihrer Definition in der Datei bis zum Ende der Datei und damit auch innerhalb der inkludierten Darei sichtbar und gültig sind, sind die Argumen-

© Copyright BISS GmbH, 2001-2012 All Rights reserved

te im zweiten Beispiel ausschließlich innerhalb der inkludierten Datei sichtbar. Nach der Zeile „`</bw:include>`“ sind beide Argumentenvariablen undefiniert.

## 3.9 <bw:lookup>

In diesem Abschnitt werden die folgenden Tags behandelt, die gemeinsam Lookup-Tables definieren:

```
<bw:lookup>
<bw:lutab>
<bw:te>
```

Lookup-Tables bieten eine einfache Möglichkeit, Wertetransformationen durchzuführen, d.h. anhand eines Wertes (im Folgenden *Schlüssel* bzw. *Key* genannt) einen anderen Wert (im Folgenden bzw. *Wert* bzw. *Value* genannt) nachzuschlagen. Beispielsweise können mit Hilfe von Lookup-Tables Status-Codes in entsprechende GIF-Images übersetzt oder die Hintergrundfarbe anhand der Zeilposition variiert werden. Lookup-Tables können in vielen Fällen umständliche IF-Konstrukte ersetzen, insbesondere können Sie innerhalb von Ausdrücken (d.h. auch innerhalb von HTML-Tag-Attributen) verwendet werden.

Die Definition einer Lookup-Table erfolgt in zwei Schritten:

1. Zunächst wird mit Hilfe des `<bw:lutab>`-Tags sowie eines oder mehrerer darin enthaltener `<bw:te>`-Tags die eigentliche Tabelle definiert.
2. Anschließend wird mit Hilfe des `<bw:lookup>`-Tags der Zugriff auf die Tabelle geschaffen.

Die Verwendung einer Lookup-Tabelle erfolgt innerhalb eines BML-Ausdrucks (siehe hierzu weiter unten sowie den Abschnitt 4.7).

### Syntax

Die Syntax der Lookup-Tags ist wie folgt:

```
<bw:lutab name="tabname">
  <bw:te key="code1" val1="wert1a" val2="wert2a" />
  <bw:te key="code2" val1="wert1b" val2="wert2b" />
  <bw:te key="code3" val1="wert1c" val2="wert2c" />
</bw:lutab>
<bw:lookup name="lookuptabname" element="te" inlist="tabname" key="@key"
/>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

<bw:lutab> hat als einziges Attribut den Namen der Lookup-Tabelle, der später innerhalb des <bw:lookup>-Tags benötigt wird (der Zugriff auf die Tabelle erfolgt nicht mit Hilfe dieses Namens sondern mit Hilfe des in <bw:lookup> definierten Namens !).

| Attribut | Bedeutung |
|---|---|
| name | Definiert den Namen der Lookup-Tabelle für die spätere Referenzierung innerhalb des <bw:lookup>-Tags |

Die einzelnen Elemente der Lookup-Tabelle werden mit Hilfe der <bw:te>-Tags definiert. Das <bw:te>-Tag stellt quasi eine Tabellenzeile dar. Diese haben immer ein "key"-Attribut, welches den Schlüsselbegriff (Key) definiert sowie ein oder mehrere (bis zu 3) Werteattribute (Values), welche die dem Schlüssel zugeordneten Werte definieren.

| Attribut | Bedeutung |
|---|---|
| key | Definiert den Schlüsselbegriff, mit dessen Hilfe auf die Werte (val1, val2, val3) zugegriffen werden kann |
| val1 | Definiert den ersten, dem Schlüssel zugeordneten Wert |
| val2 | (Optional) Definiert den zweiten, dem Schlüssel zugeordneten Wert |
| val3 | (Optional) Definiert den dritten, dem Schlüssel zugeordneten Wert |

Das <bw:lookup>-Tags definiert nun den Zugriff auf die Tabelle mit Hilfe der folgenden Attribute:

| Attribut | Bedeutung |
|---|---|
| name | Definiert denjenigen Namen, mit dessen Hilfe auf die entsprechende Lookup-Tabelle zugegriffen werden kann (dies ist der "nach aussen" sichtbare Name der Tabelle) |
| element | (Optional) Der Namen desjenigen Elements, das die einzelnen Lookup-Werte enthält; dieses Attribut ist optional, da es in der aktuellen Version der BML immer das <bw:te>-Element sein muss (bitte beachten Sie, dass der Namespace hier weggelassen wird !) |
| inlist | Referenziert den Namen der Lookup-Tabelle (siehe auch "name"-Attribut des <bw:lutab>-Tags) |
| key | (Optional) Definiert den Namen desjenigen Attributs, welches den Schlüsselbegriff enthält; auch dieses Attribut ist optional, da es in der aktuellen Version der BML immer das Attribut "@key" sein muss, welches den Schlüsselbegriff enthält |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Die Lookup-Tabelle kann anschließend innerhalb von BML-Ausdrücken verwendet werden. Die Syntax hierfür sieht wie folgt aus:

```
#lookuptabname[expr].@werteattribut
```

"`lookuptabname`" ist derjenige Name der innerhalb des Attributs "`name`" des `<bw:lookup>`-Tags definiert wurde; "`expr`" bezeichnet einen Ausdruck zur Auswahl des gewünschten Schlüsselwertes (Keys); "`werteattribut`" ist eines der innerhalb des `<bw:te>`-Tags vorkommenden Werteattribute "`val1`", "`val2`" oder "`val3`". Bitte beachten Sie, dass vor dem Tabellenname ein Doppelkreuz ('#') steht ! Selbstverständlich sollte der Schlüsselwert in der Tabelle vorkommen, sonst wird als Ergebnis ein Leerstring ausgegeben.

### Beispiel

```
<bw:lutab name="color">
  <bw:te key="0" val1="background-color:#D9D9D9;" />
  <bw:te key="1" val1="background-color:#FFFFFF;" />
</bw:lutab>

<bw:lookup name='bgcolor' element='te' inlist='color' key='@key' />
```

Der dargestellte Code definiert die folgende Tabelle:

| Key | Value |
|---|---|
| 0 | background-color:#D9D9D9; |
| 1 | background-color:#FFFFFF; |

Auf die im obigen Beispiel definierte Lookup-Tabelle kann nun wie folgt zugegriffen werden (der entsprechende BML-Ausdruck ist fett markiert):

```
<bw:foreach element="Adresse" inlist "AdresseListe">
      <td style="{#bgcolor[pos()%2].@val1}">
      <!-- Weiterer XHTML-Code -->
</bw:foreach>
```

Im Beispiel wird die jeweilige Position (`pos()`) während der Listen-Iteration verwendet, um alternierend (mit Hilfe einer Modulo-Division) zwischen den definierten Farben (weiß und grau) zu wechseln. Beachten Sie bitte, dass der Ausdruck in geschweiften Klammern stehen muss (siehe auch Abschnitt 4) !


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 3.10 <bw:page>

`<bw:page>` dient der Definition einer abgeleiteten Seite und ist näher im Abschnitt 5 (BML-Inheritance) beschrieben.

`<bw:page>` erlaubt lediglich einfache Ableitungen (single inheritance); für Mehrfachableiungen (multiple inheritance) muss das Konstrukt `<bw:include>` verwendet werden (siehe auch Abschnitt 5, *BML-Inheritance*).

### Syntax

Die Syntax des `<bw:page>`-Tags ist wie folgt:

```
<bw:page extends="url">
  … BML/XHTML-Code…
</bw:page>
```

Die Attribute haben folgende Bedeutung:

| Attribut | Bedeutung |
|---|---|
| `extends` | URL (z.B. der Dateiname) derjenigen Seite, die als Basis der abgeleiteten Seite dienen soll |

### Beispiel

```
<?xml version="1.0" encoding="iso-8859-1"?>

<bw:page extends="BWBase.BML">
  <!-- Page-spezifische Styles -->
  <bw:element mode="insert" path="html.head">
    <style>
      @import url( "styles/bwportal.css");
    </style>
  </bw:element>

  <!-- Navigation -->
  <bw:element mode="replace" id="#navigation">
    <div id="navigation">
      <p class=header"><bw:var expr="$BWNP_I_Title" /></p>
      <ul>
      <bw:foreach element="SysCmds.*Command">
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
          <li>
            <a href="{cmd( .Name)}" class="menue">
              <bw:var expr=".Label" />
            </a>
          </li>
        </bw:foreach>
        </ul>
      </div>
  </bw:element>
</bw:page>
```

## 3.11 <bw:script>

Das `<bw:script>`-Tag ermöglicht die Definition von JavaScript-Funktionen, die zum Zeitpunkt der Transformation ausgeführt werden können. Die innerhalb dieses Tags definierten JavaScript-Funktionen werden *nicht* in die Ausgabe übernommen, stehen also *nicht* während der Anzeige zur Verfügung.

Das `<bw:script>`-Tag entspricht ansonsten dem aus HTML bekannten `<bw:script>`-Tag, verwendet somit die folgende Syntax:

```
<bw:script language="JavaScript" type="text/javascript">
   <!-- ... JavaScript-Funktionen -->
</bw:script>
```

Die so definierten Funktionen können anstatt von Java-Funktionsaufrufen innerhalb von BML-Ausdrucken verwendet werden (siehe auch Abschnitt 4.6). beim Funktionsaufruf muss dem Funktionsnamen der Pärfix `"javascript:"` vorangestellt werden.

**Beispiel:**

Das folgende Beispiel definiert eine JavaScript-Funktion und zeigt deren Aufruf-Syntax.

```
<bw:script language="JavaScript" type="text/javascript">

function summeTotal ( vertraegeAcc)
{
  var ev = bsf.lookupBean( "bissBML");
  var vertraege = ev.getItem( null, vertraegeAcc);
  var bos = ev.getChilds( vertraege, "BusinessObject");
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  if ( bos == null )
    return "";

  var n = bos.size();
  var sum = 0;
  for ( i = 0; i &lt; n; i++ ) {
    var bo = bos.get( i);
    var bjb = ev.getItem( bo, "BOVersion.Bruttojahresbeitrag",
                                  true);
    var num = parseFloat( bjb);
    if ( !isNaN( num) ) {
      sum += num;
    }
  }

  return sum.toString();
}
</bw:script>

<bw:var expr="javascript:summeTotal( 'Partner.Vertraege')">
```

Bezüglich des Zugriffs auf das BISS-Objektmodell aus JavaScript heraus siehe Abschnitt 7.

## 3.12 <bw:switch>

Das `<bw:switch>`-Tag ermöglicht die einfache Definition einer Fallunterscheidung inkl. eines Standard-Verhaltens. Es ist erleichtert damit die Definition von Fallunterscheidungen, die alternativ mit Hilfe mehrerer `<bw:if>`-Tages definiert werden müssten.

**Syntax**

```
<bw:switch expr="ausdruck">
  <bw:case value="wert1">
    ... BML/XHTML-Code ...
  </bw:case>
  <bw:case value="wert2">
    ... BML/XHTML-Code ...
  </bw:case>
  <bw:case value="wert3">
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    ... BML/XHTML-Code ...
  </bw:case>
  <bw:default>
    ... BML/XHTML-Code ...
  </bw:default>
</bw:switch>
```

Ein `<bw:case>` ist immer mit einem impliziten „break“ abgeschlossen, d.h. es wird immer nur eine der genannten Alternativen verarbeitet. In der aktuellen BML-Version (4.03) ist lediglich die Angabe eines Wertes möglich (diese Einschränkung wird in der kommenden Version aufgehoben).

| Attribut | Bedeutung |
|---|---|
| `expr` | Gibt den zu evaluierenden Ausdruck an |
| `value` | Gibt einen der möglichen Werte an, zu denen `ausdruck` evaluiert werden kann. |

**Beispiel**

```
<bw:switch expr="language">
  <bw:case value="DE">
    ... BML/XHTML-Code für die deutsche Version ...
  </bw:case>
  <bw:case value="EN">
    ... BML/XHTML-Code für die englische Version ...
  </bw:case>
  <bw:case value="CZ">
    ... BML/XHTML-Code für die tschechische Version ...
  </bw:case>
  <bw:default>
    ... BML/XHTML-Code für alle anderen Sprachversionen ...
  </bw:default>
</bw:switch>
```

## 3.13 <bw:usecls>

Neben eingebauten Funktionen (siehe Abschnitt 4.6) können beliebige andere Funktionen in Java implementiert und innerhalb der BML zur Verfügung gestellt werden. Alle Funktionen, die innerhalb einer BML-Templatedatei verwendet werden sollen, müssen innerhalb einer (nicht zwingend einer einzigen !) Java-Klasse enthalten sein. Es ist auch möglich (und sinn-


© Copyright BISS GmbH, 2001-2012 All Rights reserved

voll), eine Klassenhierarchie zu entwickeln, welche an der Wurzel die generischen sowie an den Blattenden die (je BML-Templatedatei) spezifischen Funktionen enthält. Tatsächlich befindet sich an der Wurzel einer derartigen Hierarchie die Datei `biss.BML.Evaluator`, welche die BML-internen (eingebauten) Funktionen enthält.

Das BML-Tag `<bw:usecls>` gibt nun an, welche Klasse für die Suche nach Funktionen verwendet werden soll. Das `<bw:usecls>`-Tag muss zwingend vor der Verwendung der in der Datei enthaltenen Funktionen stehen. Innerhalb einer BML-Templatedatei können mehrere `<bw:usecls>`-Tags verwendet werden; die Gültigkeit eines derartigen Tags geht dabei bis zum Nächsten (siehe auch das Beispiel weiter unten) bzw. (falls kein Weiteres auftaucht) bis zum Ende der Templatedatei.

### Syntax

Das Tag hat die folgende Syntax:

```
<bw:usecls name="package.clsname" />
```

Das Tag hat ein einziges Attribut, das den Namen der Klasse (inklusive Package-Namen !) angibt. Selbstverständlich können auch alle Funktionen der Basisklassen verwendet werden.

| Attribut | Bedeutung |
|---|---|
| `name` | Name der Klasse (mit Package-Namen), in der sich die zu nutzenden Funktionen befinden |

### Beispiel

```
<bw:usecls name="biss.wise.BML.MyClass" />

<!-- Die folgende Zeile verwendet z.B. eine Funktion aus "MyClass" -->
<bw:var name="v1" expr="fullName( 'Partner')" />

<bw:usecls name="biss.wise.BML.MyOtherClass" />

<!-- Die folgende Zeile verwendet z.B. eine Funktion aus "MyOtherClass"
-->
<bw:var name="v1" expr="summeLeben( 'Partner.Vertraege')" />
```

Das obige Beispiel verwendet die Java-Klassen "`biss.wise.BML.MyClass`" (mit der Funktion "`fullName()`" sowie "`biss.wise.BML.MyOtherClass`" (mit der Funk-

© Copyright BISS GmbH, 2001-2012 All Rights reserved

tion "summeLeben()". Bis zum zweiten <bw:usecls>-Tag wird bei jedem Funktionsaufruf die Java-Klasse "MyClass" referenziert, danach die Klasse "MyOtherClass".

Ab der Version 1.4 ist es möglich, innerhalb eines Ausdrucks (Expression) einen temporären spezifischen Evaluator zur Methoden-Auswertung heraunzuziehen. Dies erfolgt mit Hilfe der folgenden Syntax:

```
java:packageName.className.MethodName(...)
```

**Beispiel**

```
java:biss.wise.BML.Evaluator.fullName('Partner')
```

## 3.14 <bw:var>

Einer der wichtigsten BML-Tags ist der <bw:var>-Tag, der einerseits dem Zugriff auf das *B-Wise* Objektmodell (DOM) dient, andererseits die Definition (Deklaration und Zuweisung) von lokalen Variablen erlaubt.

Eine der Hauptaufgaben von BML ist die Verbindung des *B-Wise* Objektmodells (DOM) mit einer XHTML-Seite zur Anzeige der DOM-Werte. Zum Zeitpunkt des Designs bzw. der Entwicklung kann es jedoch von großem Vorteil sein, die Implementierung des DOMs (Java-Codierung) vom Design der Anzeigeseiten (Seiten-Layout und XHTML-Codierung) zu entkoppeln bzw. mit Hilfe einer klaren Schnittstellenspezifikation miteinander zu verbinden. Zur Spezifikation wird eine XML-Datei verwendet, die ein Abbild des DOMs enthalten muss. <bw:var> kann daher sowohl auf das interne DOM (Speicherstruktur) als auch auf eine externe XML-Datei zugreifen.

**Syntax**

Die Syntax des <bw:var>-Tags unterstützt eine Reihe von Verwendungsmöglichkeiten. Die folgende Aufstellung gibt die alternativen Verwendungsmöglichkeiten wieder:

```
<bw:var class="stylecls" expr="ausdruck" />
<bw:var name="varname" expr="ausdruck" />
<bw:var class="stylecls" expr="ausdruck">XHTML-Code</bw:var>
```

Die erste Variante wertet den Ausdruck "ausdruck" aus und fügt das Ergebnis in die Ausgabe ein. Die zweite Variante weist dem Variablen-Namen "varname" den Wert des Ausdrucks "ausdruck" zu (eine derartig definierte Variable wird innerhalb von BML als


© Copyright BISS GmbH, 2001-2012 All Rights reserved

"interne Variable" bezeichnet im Ggs. zu externen Variablen, die den Zugriff auf das *B-Wise* Objektmodell erlauben; siehe hierzu auch Abschnitt 4.2). Die so definierte Variable kann in der Folge mit einem vorangestellten Dollarzeichen (also "`$varname`") verwendet werden. In dieser Variante findet keine Übernahme des Ergebnisses in die Ausgabe statt ! Die dritte Variante entspricht der ersten, allerdings wurde hier beliebiger XHTML-Code in den `<bw:var>`-Block eingefügt; bei der Auswertung des Ausdrucks "`ausdruck`" und der anschließenden Ausgabe der Ergebnisse wird dieser XHTML-Code nicht mit übernommen. Diese Variante ist sinnvoll, wenn die Anzeigeseite initial ohne die entsprechende XML-Datei entworfen wird und entsprechende, prototypische Ausgaben benötigt werden. Die Angabe einer Style-Klasse über das class-Attribut ist optional. Wird eine Style-Klasse auf diese Art und Weise angegeben, wird der `<bw:var>`-Ausdruck wie folgt expandiert:

```
<bw:span class="stylecls">Evaluierter var-Ausdruck ‘ausdruck’</bw:span>
```

**Hinweis:** Weitere Informationen zu Variablen enthält der Abschnitt 4.2 und 4.3.

Die Attribute sind wie folgt definiert

| Attribut | Bedeutung |
|---|---|
| `name` | (Optional) Name der Variablen; wird dieses Attribut verwendet, wird die Übernahme des Ergebnisses in die Ausgabe unterdrückt ! |
| `expr` | Ausdruck, dessen Ergebnis entweder dem Variablennamen zugewiesen oder in die Ausgabe übernommen wird. |
| `class` | (Optional) Name einer Style-Klasse (css), welche die Darstellung des Ausdrucks bestimmt (siehe auch oben). |

Auf mit Hilfe eines Namens definierte Variablen kann innerhalb der BML-Ausdrücke zugegriffen werden: dem Variablennamen muss dabei ein Dollarzeichen ('`$`') vorangestellt werden (siehe Beispiele).

Variablen sind innerhalb der BML immer global definiert, d.h. innerhalb der gesamten Datei sichtbar.

Variablen sind innerhalb der BML nicht typisiert. Innerhalb von Ausdrücken bzw. Funktionsaufrufen findet eine Typwandlung in den Typ "String" statt. Bei Infix-Operatoren wird versucht, die Argumente in den entsprechenden Typ (z.B. numerisch) zu wandeln, um die gewünschte Operation auszuführen.

So stellt das folgende Konstrukt eine Addition dar:

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:var name="str1" expr="2" />
<bw:var name="str2" expr="3" />
<bw:var expr="$str1 + $str2" />
<!-- Ergibt: 5 -->
```

### Beispiel

```
<bw:var expr="Kunde.Name" />
<bw:var name="col" expr="#bgcolor[pos()%2].@val1" />
<bw:var expr="Kunde.Name">Mustermann</bw:var>
<bw:var class="mycls" expr="Kunde.Name">Mustermann</bw:var>
```

Die erste Zeile wertet den Ausdruck "`Kunde.Name`" aus und fügt das entsprechende Ergebnis in die Ausgabe ein. Die zweite Zeile weist der Variablen mit dem Namen "`col`" einen Wert aus der Lookup-Tabelle zu (siehe auch 3.9 zu `<bw:lookup>`). Die dritte Zeile wertet den Ausdruck "`Kunde.Name`" aus und fügt das entsprechende Ergebnis in die Ausgabe ein; ist keine entsprechende XML-Datei oder kein Zugriff auf das DOM vorhanden, wird der Konstante Texte "`Mustermann`" ausgegeben. Bei Zugriff auf das DOM bzw. Vorhandensein einer entsprechenden XML-Datei wird dieser Text unterdrückt.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Kapitel 4

# BML-Ausdrücke

Ausdrücke stellen eines der mächtigsten Funktionalitäten innerhalb der BML dar. Ausdrücke können sehr einfach sein aber auch komplexe Pfade mit Prädikaten annehmen, so dass viele Sachverhalte mit Hilfe eines Ausdrucks definiert werden können.

Ein Ausdruck ist eine Formel, die im Verlauf der Transformation ausgewertet wird und ein Resultat ergibt. Ausdrücke kommen immer nur in Attributen vor. Dabei können Ausdrücke sowohl in Attributen von BML-Tags vorkommen als auch in Attributen von XHTML-Tags; in Letzteren müssen sie in geschweiften Klammern ('{' und '}') stehen. Ausdrücke können Konstanten (Literale), interne Variablen, externe Variablen, Operatoren, Prädikate, Funktionsaufrufe und Lookup-Table-Referenzen enthalten.

**Beispiele:**

```
<img src="mybitmap.gif" class="{$imgclass}" />

<a href="#" class="menue2" title="Hostadresse anzeigen"
     onClick="showPopUp( event, 'Abweichende Hostadresse',
'{$hostadr}')">
```

## 4.1 Konstanten

Eine Konstante bzw. ein Literal ist eine Zahl, ein String oder der Wert 'true' bzw. 'false'. Stringkonstanten müssen bei der Verwendung in einfache Hochkommata eingeschlossen sein. Die folgende Aufstellung gibt einige Beispiele für erlaubte, gültige Konstanten:

```
'Test'
23
45.50
true
false
'Dies ist ein &amp;Test&amp;'
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Wie dem Beispiel zu entnehmen ist, werden Sonderzeichen mit Hilfe der XHTML-Entitätennamen (z.B. '`&`') dargestellt. "true" und "false" sind spezielle Konstanten, die ohne Anführungszeichen verwendet werden.

## 4.2 Interne Variablen

Interne Variablen sind mit Hilfe des `<bw:var>`-Tags definierte Variablen, die innerhalb der BML-Templatedatei verwendet werden können. Ein BML-Variablenname kann aus Buchstaben, Ziffern sowie dem Unterstrich bestehen (z.B. "`col_01`").

Bei der Verwendung (nicht bei der Definition) wird dem Variablennamen ein Dollarzeichen ('`$`') vorangestellt. Beispiele für interne Variablen sind:

```
<bw:var name="col" expr="#FFFFFF" />
<td style="background-color: {$col}">
```

## 4.3 Externe Variablen

Externe Variablen sind Referenzen auf Elemente innerhalb des DOMs bzw. der XML-Datei. Hierarchien (bzw. Pfade) werden dabei mit Hilfe der Punktnotation abgebildet (die einzelnen Elementnamen sind durch einen Punkt getrennt). Beispiele:

```
Kunde.Name
Partner.AdresseListe.Adresse.Ort
Partner.AdresseListe.Adresse.@id
```

Bei Verwendung einer XML-Datei ist es wichtig, zwischen Elementen und Attributen zu unterscheiden. Attribute werden dabei (analog XSL) mit einem voranstehenden '`@`' dargestellt, z.B. "`@id`". Elementnamen werden für sich stehend verwendet. Diese Unterscheidung ist bei Verwendung des DOM nicht erforderlich (innerhalb des DOMs existiert diese Unterscheidung nicht). Ein @ bei Attributen ist jedoch auch bei Zugriff auf das DOM nicht störend, da es vom Transformator automatisch vernachlässigt wird.

## 4.4 Operatoren

BML stellt eine Reihe von Operatoren zur Verfügung. Die folgende Aufstellung gliedert sich nach arithmetischen, Vergleichs-, logischen und Stringoperatoren.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Arithmetische Operatoren

| Operator | Bedeutung |
|---|---|
| `+` | Addition; bildet die Summe der beiden Operanden (Op) (`expr1 + expr2`) |
| `-` | Subtraktion; bildet die Differenz von Op_1 und Op_2 (`expr1 - expr2`) |
| `*` | Multiplikation; bildet das Produkt von Op_1 und Op_2 (`expr1 * expr2`) |
| `/` | Division; bildet den Quotienten von Op_1 und Op_2 (`expr1 / expr2`) |
| `div` | Division; entspricht dem '/'-Symbol |
| `%` | Modulo-2-Division; gibt den (Integer-) Rest der Division von Op_1 und Op_2 zurück (`expr1 % expr2`), z.B. ergibt `3 % 2 -> 1;` siehe auch Funktion `"mod2()"` |

### Vergleichsoperatoren

| Operator | Bedeutung |
|---|---|
| `=` | Test auf Gleichheit; ergibt `true` (wahr), wenn beide Operanden identisch sind |
| `!=` | Test auf Ungleichheit; ergibt `true` (wahr), wenn sich beide Operanden unterscheiden |
| `le` | kleiner oder gleich (less equal); ergibt `true` (wahr), wenn Operand 1 kleiner oder gleich Operand 2 ist |
| `lt` | kleiner als (less than); ergibt `true` (wahr), wenn Operand 1 kleiner Operand 2 ist |
| `ge` | größer oder gleich (greater equal); ergibt `true` (wahr), wenn Operand 1 größer oder gleich Operand 2 ist |
| `gt` | größer als (greater than); ergibt `true` (wahr), wenn Operand 1 größer Operand 2 ist |
| `ne` | ungleich (not equal); ergibt `true` (wahr), wenn Operand 1 ungleich Operand 2 ist (entspricht dem Symbol '!=') |
| `eq` | gleich (equal); ergibt `true` (wahr), wenn Operand 1 und Operand 2 identisch sind (entspricht dem Symbol '=') |

### Logische Operatoren

| Operator | Bedeutung |
|---|---|
| `and` | Logisches UND; das Ergebnis des Ausdrucks ist `true` (wahr) wenn sowohl Operand 1 als auch Operand 2 wahr sind |
| `or` | Logisches ODER; das Ergebnis des Ausdrucks ist `true` (wahr), wenn Operand 1 oder Operand 2 oder beide Operanden wahr sind |
| `not` | Logisches NICHT; das Ergebnis ist wahr, wenn der Operand `false` (falsch) ist |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Beispiele für die Verwendung von Operatoren**

```
<bw:var expr="Partner.Alter + 5" />
<bw:var name="col" expr="#bgcolor[pos()%2].@val1" />
<bw:if expr="Partner.Alter lt 65"> ... </bw:if>
```

# 4.5 Prädikate

Prädikate können vereinfacht als Filter (meist in Pfadangaben) verstanden werden, die bestimmten, ob das Ergebnis eines Teilausdrucks in Betracht gezogen werden soll oder nicht. Betrachten wir hierzu am besten ein einfaches Beispiel:

```
<bw:var name="col" expr="Kunde.AdresseListe.*Adresse.Ort" />
```

Hier handelt es sich um eine typische Pfadangabe, d.h. die Angabe desjenigen Pfads, den man innerhalb der Elemente-Hierarchie (angefangen beim "Kunden") durchwandern muss, um zur gewünschten Endvariablen - hier "Ort" - zu gelangen. Bitte beachten Sie, dass innerhalb von Pfadangaben, welche Listen bzw. Listenelemente enthalten dem Namen des Listenelements ein Sternchen ('*') vorangestellt werden muss.

Da wir innerhalb der "AdresseListe" jedoch zwei "Adresse"-Elemente haben, ist das Ergebnis des Ausdrucks unbestimmt (BML verwendet normalerweise immer das physisch erste Element). Möchten wir nun gezielt ein bestimmtes Adress-Element herausgreifen, benötigen wir zusätzliche Angaben (oder einen "Filter"), um das gewünschte Element präzise bestimmen (aus der Liste der Elemente "herausfiltern") zu können. Diese Angaben können wir mit Hilfe von Prädikaten definieren:

```
<bw:var name="col" expr="Kunde.AdresseListe.*Adresse[@typ=2].Ort" />
```

Hier wird dasjenige Element "Adresse" herausgefiltert, dessen Attribut "typ" den Wert '2' besitzt.

Wir können den Gesamtausdruck wie folgt lesen (siehe hierzu auch die Beispiel-XML-Datei des Abschnitts 2): Der Wert des (Kind-)Elements "Ort" desjenigen (Eltern-)Elements "Adresse", dessen Attribut "typ" den Wert '2' hat, der Liste "AdresseListe" des Elements "Kunde". Oder andersherum: Ausgehend vom (hierarchisch obersten) Element "Kunde" verwende das (Kind-)Element "AdresseListe", anschließend dasjenige (Kind)Element "Adresse", dessen Attribut "typ" den Wert '2' besitzt und verwende schließlich den Wert des (Kind-)Elements "Ort".


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Innerhalb von Prädikatausdrücken können mit Ausnahme anderer Prädikate alle o.g. Anteile (also Konstanten, interne und externe Variablen, Operatoren und Funktionen) vorkommen. Prädikate können innerhalb einer Pfadangabe an mehrere Elemente geknüpft werden. Auch hierzu ein Beispiel:

```
<bw:var name="T2"
   expr="Kunde[@id='23' and GebDat='23.06.1955'].AdrList.*Adr[id=1].Ort"
/>
```

Hier wird der Wert des (Sub-)Elements "`Ort`" ausgegeben, dessen Elternelement "`Adr`" ein (Sub-)Element "`id`" mit dem Wert 1 besitzt und zur Liste "`AdrList`" gehört, allerdings nur wenn die Liste zum Element "`Kunde`" mit einem Attribut "`id`" mit dem Wert "`23`" und einem (Sub-)Element "`GebDat`" mit dem Wert "`23.06.1955`" gehört.

# 4.6 Funktionen

BML verfügt über einige eingebaute Funktionen. Zusätzlich zu diesen können beliebige Funktionen innerhalb einer assoziierten Java-Klasse definiert und aus der BML heraus aufgerufen werden (siehe hierzu Abschnitt 6, "BML und Java").

Folgende Funktionen sind in der BML standardmäßig enthalten (diese Funktionen stammen aus den Klassen „com.biss.path.Evaluator“ und „com.biss.bml.Evaluator“):

### Stringfunktionen

| Funktion | Bedeutung |
|---|---|
| **`amp( str)`** | Liefert den als Argument übergebenen String mit einem vorangestellten Ampersand (&) zurück |
| **`and( str1, str2)`** | Liefert „true“ zurück, falls str1 und str2 nicht leer sind |
| **`beginTag( str)`** | Liefert den als Argument übergebenen String in Form eines XML-Tags, also innerhalb der beiden Zeichen „<“ und „>“ |
| **`bit( str1, str2)`** | Liefert den binären Wert der str2-ten Stelle (0 oder 1) der Zahl str1; beide Argumente müssen Integer-Werte sein, str2 muss zwischen 0 und 31 liegen |
| **`concat( str1, ..., str6)`** | Verknüpft zwei bis max. 6 Strings miteinander; z.B. `concat( 'hello ', 'world')` verknüpft die beiden Strings 'hello ' und 'world' zum Ergebnis 'hello world'; es werden keine Zeichen (auch keine Leerzeichen) eingefügt. Die Funktion liefert immer `true` zurück. |
| **`concatNe( str1, ..., str5)`** | Verknüpft zwei bis max. 5 Strings miteinander, jedoch nur |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | dann, wenn keiner der Strings ein Leerstring ist; z.B. `concat( 'hello ', 'world')` verknüpft die beiden Strings 'hello ' und 'world' zum Ergebnis 'hello world'; es werden keine Zeichen (auch keine Leerzeichen) eingefügt. Ist einer der Strings ein Leerstring, wird als Ergebnis ein Leerstring ausgegeben. Diese Funktion ist nützlich, wenn konstante Zeichenfolgen (z.B. ein '*' oder Klammern) nur dann ausgegeben werden sollen, wenn der variable Anteil vorhanden ist. Die Funktion liefert immer `true` zurück. |
| **singleQuote()** | Liefert ein einzelnes, einfaches Anführungszeichen (') |
| **singleQuote( str)** | Liefert den String `str` innerhalb von einfachen Anführungszeichen. Beispiel: `singleQuote( Hallo)` liefert `'Hallo'`. |
| **doubleQuote()** | Liefert ein einzelnes doppeltes Anführungszeichen ("); Hinweis: doppelte Anführungszeichen können auch als Entitäten mit Hilfe des Ausdrucks `"` dargestellt werden. |
| **doubleQuote( str)** | Liefert den String `str` innerhalb doppelter Anführungszeichen. Beispiel: `doubleQuote( Hallo)` liefert `"Hallo"`. |
| **formatDate( str1, str2)** | Formatiert str1 gemäß der Anweisung in str2 als Datum |
| **formatNumber( str1, str2)** | Formatiert str1 gemäß der Anweisung in str2 als Zahl |
| **formatTimestamp( str1, str2)** | Formatiert str1 gemäß der Anweisung in str2 als Zeitstempel |
| **iff( str1, str2, str3)** | Wertet die in str1 enthaltene Bedingung und gibt str2 zurück, falls die Bedingung wahr ergibt, ansonsten str3 |
| **indexOf( str1, str2)** | Gibt den numerischen, 0-basierten Index aus str1, an dem str2 gefunden wurde, ansonsten -1 (mappt auf indexOf in Java) |
| **inn( str1, str[])** | Gibt true zurück, falls str1 im String-Array str[] vorkommt, ansonsten false |
| **join( delim, str1, ..., strN)** | Verknüpft beliebig viele Strings (`str1` bis max. `strN`) miteinander und fügt zwischen diese den String `'delim'`, z.B.: `join( ', ', 'hello', 'world')` verknüpft die beiden Strings 'hello' und 'world' zu 'hello, world'. Die Funktion liefert immer `true` zurück. Leere Argumente werden bei der Verknüpfung ausgelassen. |
| **lastIndexof( str1, str2)** | Gibt den letzten 0-basierten Index von str2 ind str1 zurück oder -1 falls str2 nicht in str1 vorkommt |
| **compare( str1, str2)** | Vergleicht die beiden Strings `str1` und `str2` auf Gleichheit und liefert 0 zurück, falls beiden Strings gleich sind, -1 falls `str1` lexikographisch kleiner ist als `str2`, andernfalls 1. |
| **encloseIf( str1, str2,** | Liefert str3 eingebettet in str2 und str4, falls str1 nicht leer ist, |



© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| **str3, str4)** | ansonsten wird str3 unverändert zurückgegeben |
| **endsWith( str1, str2)** | Liefert `true` zurück, falls `str1` mit `str2` endet, andernfalls false, z.B. `endsWith( 'hello world', 'ld')` liefert true zurück. |
| **endTag( str)** | Liefert den als Argument übergebenen String in Form eines schließenden XML-Elements zurück (</str>). |
| **delim( str)** | Liefert '`str`' zurück, falls '`pos() > 0`' ansonsten einen Leerstring (''). Diese Funktion ist hilfreich, wenn ein Trennzeichen zwischen die String einer Liste eingefügt werden soll, jedoch nicht am Ende der Ausgabe (Liste) erscheinen soll. |
| **startsWith( str1, str2)** | Liefert `true`, falls `str1` mit `str2` beginnt, z.B. `startsWith( 'hello world', 'hell')` liefert `true`. |
| **substring( str1, str2, str3)** | Liefert aus `str1` den Substring von der (0-basierten) Position `str2` bis zur (0-basierten) Position `str3 -` 1, z.B. liefert `substring( 'Hello world', '6', '9')` den Wert "`wor`" zurück. |

### Arithmetische Funktionen

| Funktion | Bedeutung |
|---|---|
| **ceil( n1)** | Berechnet die kleinste Ganze Zahl die größer oder gleich der als Argument übergebenen Zahl ist. |
| **count( expr)** | Liefert die Anzahl der dem Ausdruck expr genügenden Elemente; Beispiel siehe Anhang D. Diese Funktion wird in Zusammenhang mit Pfadausdrücken und Listen verwendet. |
| **div( n1, n2)** | Dividiert die Zahl n1 durch die Zahl n2 |
| **floor( n1)** | Gibt denjenigen Wert zurück der kleiner oder gleich dem übergebenen Argument ist |
| **mod( n1, n2)** | Liefert den (Integer-) Rest der Division von `n1` durch `n2`, z.B. `mod( 23, 3)` liefert den Wert 2. |
| **mod2( n1)** | Liefert den (Integer-) Rest der Division von `n1` durch 2, z.B. `mod2( 5)` liefert den Wert 1; diese Funktion entspricht dem Infix-Operator '`%`'. |
| **sgn( n1)** | (Signum-Funktion) Liefert folgende Werte:<br>0 falls `n1` gleich 0<br>-1 falls `n1` kleiner als 0<br>1 falls `n1` größer als 0 |
| **sum( expr)** | Summiert alle Elemente innerhalb der Expression "`expr`" auf; Beispiel siehe Anhang D. Diese Funktion wird in Zusammenhang mit Pfadausdrücken und Listen verwendet. |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Command-Funktionen

Eine herausragende Bedeutung innerhalb der BML haben die "`cmd`"-Funktionen. Sie ermöglichen innerhalb von Hyperlinks (also `<a href="..."> ... </a>`-Blöcken) die Definition von Funktionsaufrufen an den *B-Wise*-Server. Damit kann der Anwender durch Click auf einen Hyperlink einen beliebigen Prozess starten bzw. eine Funktion im *B-Wise* Server aufrufen. Diese Befehle können bis zu 5 Parameter besitzen, die normalerweise aus dem DOM ausgelesen werden (z.B. "`Partner.ID`"). Ob die innerhalb der BML-Templatedatei definierten Befehle allerdings tatsächlich in die Ausgabe (HTML) übernommen werden, hängt von der Berechtigungsstufe des Anwenders ab. Diese wird innerhalb des Obketmodells (DOMs) innerhalb eines speziellen Blocks (des sog. "`Grant`"-Blocks) definiert. Nur wenn der innerhalb der BML-Templatedatei angegebene Befehl (z.B. "`editPartner`") auch innerhalb des "`Grant`"-Blocks vorkommt, wird dieser auch in die Ausgabe übernommen.

Sollte der aktuell eingeloggte Anwender keine Berechtigung zur Ausführung z.B. des "`editPartner`"-Prozesses besitzen, ist es in manchen Situationen erwünscht, den zugehörigen HTML-Text (oder ein Icon) beizubehalten, oder den gesamten Befehl samt zugehörigem Text (also Inhalt des `<a>`-Tags) nicht in die Ausgabe zu übernehmen. Dies erfolgt durch Verwendung der "`cmd`"- bzw. "`cmdIg`"-Funktion.

In der Praxis sieht das wie folgt aus:

```
<!-- Auszug aus einer XML-Datei bzw. dem Objektmodell -->
<Grant>
   <Cmd name="assignGroup"     p1="{Partner.ID}" />
   <Cmd name="deletePartner"   p1="{Partner.ID}" />
   <Cmd name="changeName"      p1="{Partner.ID}" />
   <Cmd name="createContract"  p1="{Partner.ID}" />
   <Cmd name="createPartner"   p1="{Partner.ID}" />
   <Cmd name="showPartner"     p1="{Partner.ID}" />
</Grant>

<!-- Auszug aus der BML-Templatedatei -->
<table class="tab" cellpadding="1" cellspacing="0">
   <colgroup>
     <col width="60%" />
     <col width="40%" />
  </colgroup>
   <tr>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      <td class="head1">
         <img src="toggle1.gif" onclick="toggle( doc.all.PP_B1);" />
         <a href="{cmd( 'showPartner')}" class="menue">Basisinfo</a>
      </td>
      <td class="head1" align="right">
         <a href="{cmdIg( 'printPartner')}">
            <bw:img src="edit02.gif" />
          </a>
         <a href="{cmd( 'deletePartner')}">
            <bw:img src="edit02.gif" />
          </a>
         <a href="{cmd( 'writeLetter')}">
            <bw:img src="edit02.gif" />
          </a>
         <a href="{cmd( 'searchDouble')}">
            <bw:img src="edit02.gif" />
          </a>
         <a href="{cmd( 'assignGroup')}">
            <bw:img src="edit02.gif" />
          </a>
         <a href="{cmd( 'createPartner')}">
            <bw:img src="edit02.gif" />
          </a>
      </td>
   </tr>
</table>
```

Da lediglich die fett markierten Befehle innerhalb des "Grant"-Blocks vorkommen, werden diese auch (als Hyperlinks) in die Ausgabe übernommen. Mit Ausnahme des Icons für den Befehl "`printPartner`" werden allerdings alle Icons angezeigt (diejenigen ohne Befehl haben allerdings keine Wirkung). Das Icon für den Befehl "`printPartner`" wird nicht angezeigt (nicht in die Ausgabe übernommen, da der entsprechende Hyperlink-Block die Funktion "`cmdIg`" verwendet (siehe unten).

**`cmd( name, p1, ..., p5)`**

Ersetzt alle ggf. vorhandenen Variablen und liefert anschließend den syntaktisch korrekten Wert für den Befehl `name` sowie dessen Parameter `p1` bis max. `p5`, z.B. `cmd( 'editPartner', Partner.ID)` liefert `CMD#editPartner#p1=1` falls der Wert der Variablen "`Partner.ID`" gleich 1 ist; dabei wird der Inhalt des Hyperlinks in jedem Falle in die Ausgabe übernommen.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

BISS

**`cmdIg( name, p1, ..., p5)`** Ersetzt alle ggf. vorhandenen Variablen und liefert anschließend den syntaktisch korrekten Wert für den Befehl `name` sowie dessen Parameter `p1` bis max. `p5`, z.B. `cmd( 'editPartner', Partner.ID)` liefert `CMD#editPartner#p1=1` falls der Wert der Variablen "`Partner.ID`" gleich 1 ist; dabei wird der Inhalt des Hyperlinks nur dann in die Ausgabe übernommen, falls der Befehl im "Grant"-Block vorhanden ist (siehe oben).

### Formatfunktionen

Die Formatfunktionen erlauben eine Formatierung bei der Ausgabe von Zahlen (insbesondere Währungen) sowie Tagesdaten.

Aktuell stehen zwei Formatfunktionen zur Verfügung:

- `formatDate( datestr, pattern)` und
- `formatNumber( numstr, pattern)`

**`formatDate`** formatiert ein Tagesdatum ("`datestr`") mit Hilfe folgender Patterns ("`pattern`"):

| Symbol | Form | Bedeutung |
|---|---|---|
| G | Text | Ära-Definition, z.B. "AD" |
| y | Zahl | Jahr |
| M | Text/Zahl | Monat im Jahr, z.B. "August" oder "08" |
| d | Zahl | Tag |
| h | Zahl | Stunde im 0-12 Format (also a.m. bzw. p.m.) |
| H | Zahl | Stunde im 24h-Format |
| m | Zahl | Minute |
| s | Zahl | Sekunde |
| S | Zahl | Millisekunde |
| E | Text | Wochentag, z.B. "Mittwoch" |
| D | Zahl | Tag innerhalb eines Jahres, z.B. "189" |
| F | Zahl | Wochentag innerhalb eines Monats |
| w | Zahl | Woche innerhalb eines Jahres, z.B. "48" |
| W | Zahl | Woche innerhalb eines Monats |
| a | Text | a.m./p.m.-Marker |
| k | Zahl | Stunde innerhalb eines Tages, z.B. "23" |
| ' | - | Escape-Zeichen für Text |
| " | - | Literal in einfachen Anführungszeichen |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Hinweis:** Beachten Sie, dass bei den Pattern-Zeichen zwischen Groß- und Kleinschreibung unterschieden wird.

Die Anzahl der Pattern-Zeichen bestimmt das Format:

| Typ | Hinweise |
|---|---|
| Text | 4 oder mehr Zeichen -> verwendet volle Form, weniger als 4 Zeichen -> verwendet eine abgekürzte Form (falls existent) |
| Zahl | Bestimmt die minimale Anzahl Ziffern. Kürzere Zahlen werden mit Nullen aufgefüllt. Jahresangaben werden besonders behandelt: falls die Anzahl der Patternzeichen "y" gleich zwei ist, wird das Jahr auf 2 Stellen begrenzt. |
| Text & Zahl | 3 oder mehr Zeichen -> verwendet Text, ansonsten verwendet Zahl. |

Alle Zeichen außer "a...z" bzw. "A...Z" werden als Literale behandelt, auch wenn sie nicht in einfachen Anführungszeichen stehen.

Beispiel für einen entsprechenden Funktionsaufruf:

```
formatDate( Partner.Geboren, 'yyyy.MM.dd')
```

**`formatNumber`** formatiert eine Zahl ("`numstr`") mit Hilfe folgender Patterns ("`pattern`"):

| Symbol | Bedeutung |
|---|---|
| 0 | Ziffer |
| # | Ziffer, wobei eine Null unterdrückt (nicht angezeigt) wird |
| . | (Punkt) Dezimaltrennzeichen oder Dezimaltrennzeichen für Währungsdarstellung |
| - | Minus-Symbol (Symbol für negative Werte) |
| , | Gruppenseparator (normalerweise verwendet als Tausender-Trennzeichen) |
| E | Mantisse |

Beispiel für einen entsprechenden Funktionsaufruf:

```
formatNumber( Partner.Gehalt, '#,##0.00')
<!-- Erzeugt aus "123456.77" den String "123,456.77" -->
```

**Hinweis:** Alle Funktionsparameter müssen in BML-Funktionen als Strings übergeben werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 4.7 Lookup-Tabellen

Lookup-Tabellen stellen spezielle Ausdrücke dar, die im Zusammenhang mit den Lookup-Tags (siehe Abschnitt 3.9) verwendet werden. Sie ermöglichen das einfache "Nachschlagen" (und Ersetzen) von Werten anhand eines Schlüssels. In der Praxis kommt es häufig vor, dass innerhalb der Anzeigeseite Icons (Bitmaps) für bestimmte Stati oder Codes verwendet werden sollen (z.B. ein grünes Häkchen für den Status 1 - Ok oder ein rotes Verbotssymbol für den Status 2 - Fehler). Diese Aufgabe lässt sich alternativ mit einer Reihe von IF-Konstrukten lösen oder (viel einfacher und effektiver) mit Hilfe einer Lookup-Tabelle. Letztere ist nicht nur schneller sondern vor allem übersichtlicher und damit pflegeleichter.

Die Erstellung einer Lookup-Tabelle wurde bereits im Abschnitt 3.9 ("BML-Sprachkonstrukte") ausführlich beschrieben. An dieser Stelle soll die Nutzung der Tabellen innerhalb von Ausdrücken dargestellt werden.

Um die Werte der Lookup-Tabelle nutzen zu können, benötigen wir drei Informationen:

- Welche Tabelle wollen wir verwenden (Tabellen-Namen)
- Den Schlüsselwert (Key)
- Welchen Wert der Tabelle wollen wir verwenden (Value); dies ist erforderlich, da BML-Lookup-Tabellen mehrere Werte zu einem Schlüssel enthalten und zurückgeben können

Diese Werte werden wie folgt innerhalb des Ausdrucks kombiniert::

```
{#tabellenname[key].value}
```

z.B.

```
<td style="{#bgcolor[pos()%2].@val1}">
```

Im Beispiel wird die Lookup-Tabelle "`bgcolor`" verwenden. Mit Hilfe des Ausdrucks "`pos()%2`", d.h. mit Hilfe der Modulo-2 Division der aktuellen Position (innerhalb einer Schleife), wird der Schlüsselwert (Key) bestimmt, mit dessen Hilfe auf die Tabelle zugegriffen wird. Als Rückgabe (Tabellen-Wert bzw. Value) verwenden wir das Attribut "`@val`" des entsprechenden Tabellen-Eintrags (`<bw:te>`). Eine Modulo-2 Division alterniert zwischen den Werten 0 und 1, so dass wir jeweils auf den ersten (`<bw:te key="0" ... />`) bzw. zweiten Eintrag (`<bw:te key="1" ... />`) zugreifen.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Kapitel 5

# BML-Inheritance

Die BML-Inheritance (Ableitung) erlaubt den Aufbau flexibler und damit einfach wartbarer Strukturen. Sie ist zusammen mit der Aggregation (mit Hilfe von `<bw:include>`) ein Mittel zur Strukturierung von BML-Seiten.

Für die BML-Inheritance stehen insbesondere folgende Elemente zur Verfügung:

```
<bw:page>
<bw:element>
```

Das folgende Beispiel soll die Verwendung der BML-Inheritance verdeutlichen.

Alle PopUp-Dialoge einer Anwendung sollen ein grundsätzlich ähnliches Layout haben. Um dies zu erreichen und die spätere Pflege zu erleichtern gibt es mehrere Möglichkeiten. Eine davon ist die Definition einer Basis-Seite, von der alle PopUp-Dialoge (bzw. die entsprechenden Portlets) abgeleitet sind. Eine andere wäre die Verwendung des Aggregationsmechanismus, d.h. die Auslagerung der gleich aussehenden Teile in externe (zu includierende) Dateien oder die Nutzung der BML-Regeln (Rules), die im Abschnitt 8 erläutert sind. In diesem Kapitel lösen wir das Problem mit Hilfe der BML-Inhertiance[2].

Zunächst muss die Basis-Seite (das Basis-Portlet) definiert werden. Der zugehörige Code sieht wie folgt aus (aus didaktischen Gründen wurde bewusst auf ein „Ausschmücken“ verzeichtet):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:portlet
   xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml"
   id="Test_SamplePopUpBase"
   titlebar="false"
   target="cont">
```

[2] Welcher Mechanismus letztlich innerhalb eines Systems oder eines Projekts angewendet wird, hängt von unterschiedlichen Faktoren ab. Man kann daher nicht grundsätzlich den einen oder anderen Mechansimus bevorzugen, sondern muss unter Abwägung der Vor-und Nachteile je nach Situation und Anforderungen von Fall zu Fall entscheiden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:rect id="DR_1" fill="true" zorder="10">
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
           eox="0" eoy="0" ecx="0" ecy="65" />
 <bw:style>
    <bw:backgroundcolor>white</bw:backgroundcolor>
    <bw:color>white</bw:color>
 </bw:style>
</bw:rect>
<bw:label id="LA_Head1" class="h1">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="14" eoy="5" ecx="306" ecy="27" />
 <bw:style>
    <bw:backgroundcolor>white</bw:backgroundcolor>
    <bw:font>Arial,10,Bold</bw:font>
 </bw:style>
  <bw:content><bw:var expr="DlgTitle"/></bw:content>
</bw:label>
<bw:label id="LA_Info1">
 <bw:style>
    <bw:backgroundcolor>white</bw:backgroundcolor>
 </bw:style>
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
           eox="14" eoy="29" ecx="-94" ecy="58" />
  <bw:content><bw:var expr="DlgInfoTxt"/></bw:content>
</bw:label>
<bw:img id="DI_1" src="images/DlgPopUp.gif" zorder="15">
  <bw:pos aox="1000" aoy="0" acx="1000" acy="0"
           eox="-53" eoy="5" ecx="-15" ecy="56" />
</bw:img>

<bw:compound id="CP_DlgMain" tabindex="1">
 <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
           eox="0" eoy="70" ecx="0" ecy="-45"/>
</bw:compound>

<bw:line id="DL_99">
 <bw:pos aox="0" aoy="1000" acx="1000" acy="1000"
           eox="10" eoy="-40" ecx="-10" ecy="-40"/>
</bw:line>
<bw:button id="BU_Cancel" tabindex="100">
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
   <bw:pos aox="1000" aoy="1000" acx="1000" acy="1000"
             eox="-174" eoy="-32" ecx="-94" ecy="-8"/>
   <bw:content>~Abbrechen</bw:content>
   <bw:events>
      <bw:event name="clicked" method="{event( DLG_CANCEL)}"/>
   </bw:events>
  </bw:button>
  <bw:button id="BU_Ok" tabindex="99" def="true">
   <bw:pos aox="1000" aoy="1000" acx="1000" acy="1000"
             eox="-90" eoy="-32" ecx="-10" ecy="-8"/>
   <bw:content>~Ok</bw:content>
   <bw:events>
      <bw:event name="validate" method="{event( DLG_OK)}"/>
   </bw:events>
  </bw:button>

</bw:portlet>
```

Dieser Code erzeugt folgenden Dialog:

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Abb. 13: Basis-PopUp-Dialog (Beispiel)**

Um auf dieser Basis durch Ableitung einen spezifischen Dialog (z.B. für die Darstellung einer einfachen Kundenliste) zu erzeugen, verwenden wir die beiden erstgenannten Elemente wie folgt (die Elemente sind fett hervorgehoben):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:page extends="Test_SamplePopUpBase.plt"
     xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
     xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:element id="CP_DlgMain" mode="insert">
   <bw:list id="LI_KundeKurz" tabindex="1">
     <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
            eox="10" eoy="10" ecx="-10" ecy="-10" />
     <bw:content>
       <bw:lhead>
         <bw:lc width="100%"><bw:t>Kundenname</bw:t></bw:lc>
       </bw:lhead>
       <bw:lbody>
         <bw:ldata element="">
           <bw:li id="1">
             <bw:lc>Mustermann, Max</bw:lc>
           </bw:li>
           <bw:li id="1">
             <bw:lc>Musterfrau, Petra</bw:lc>
           </bw:li>
         </bw:ldata>
       </bw:lbody>
     </bw:content>
     <bw:events>
       <bw:event name="selected" method="{DO_SEL}" />
     </bw:events>
   </bw:list>
  </bw:element>

</bw:page>
```

Dieser Code sorgt dafür, dass der Transformer zunächst die „Basis“ („`Test_SamplePopUpBase.plt`“) sucht und lädt, darin das Element mit der ID


© Copyright BISS GmbH, 2001-2012 All Rights reserved

„CP_DlgMain“ sucht und - fall dieses Element gefunden wird - den Inhalt des <bw:elements> (also die Liste) in das Element mit der ID „CP_DlgMain“ einfügt (mode=“insert“).

Neben dieser einfachen Einfügeoperation können komplexere Einfüge- oder Replace-Operationen mit Hilfe von Pfadausdrücken durchgeführt werden. So sorgt z.B. der folgende Code

```
<bw:element path="bw:label[@id='LA_Info1'].bw:style" mode="replace">
  <bw:style>
    <bw:backgroundcolor>#d9d9d9</bw:backgroundcolor>
  </bw:style>
</bw:element>
```

dafür, dass die Hintergrundfarbe des Rechtecks statt in weiß (wie im Basis-Dialog definiert) in grau dargestellt wird.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

BISS

Kapitel 6

# BML und Java

BML verfügt über eine Reihe vordefinierter (eingebauter) Funktionen, die innerhalb von Ausdrücken verwendet werden können. Es gibt jedoch Situationen, in denen eine spezielle Funktion benötigt wird. Um derartige Bedürfnisse abzudecken, kann zu jeder BML-Templatedatei eine Java-Klasse erstellt werden, welche die von dieser Templatedatei benötigten Funktionen implementiert. Auf diese Funktionen kann anschließend während der Transformation zugegriffen werden.

Dieser Abschnitt erläutert anhand eines Beispiels die Erstellung einer solchen Java-Klasse sowie die Nutzung der Funktionen aus einer BML-Templatedatei heraus.

Alle Funktionen, die innerhalb einer BML-Templatedatei verwendet werden sollen, müssen sich innerhalb einer Java-Klasse befinden. Diese Klasse kann bzw. sollte Teil einer Klassenhierarchie sein, in der die generischen Funktionen an der Wurzel sowie die spezifischen Funktionen an den Blattenden angesiedelt sind. Tatsächlich befinden sich die BML-internen (eingebauten) Funktionen innerhalb der Klasse "`biss.BML.Evaluator`", die als Basis für die Ableitung aller Klassen dient, welche BML-Funktionen enthalten sollen.

Als Beispiel sowie zur Veranschaulichung von BML-Funktionen bzw. deren Implementierung wollen wir im Folgenden die Java-Klasse "`MyClass.java`" vorstellen, mit den Funktionen "`fullName()`" sowie "`summeLeben()`" bzw. "`summeTotal()`". Die Funktion "`fullName()`" bekommt als Argument einen Objektnamen, in dem bestimmte Elemente enthalten sein müssen und gibt den vollständigen (verketteten) Namen zurück. Die Funktionen "`summeLeben()`" und "`summeTotal()`" bekommen als Argument einen Objektnamen, in dem sich bestimmte Vertragsinformationen befinden müssen und gibt die Beitragssumme aller Lebensverträge bzw. die Summe aller Beiträge zurück.

Im Folgenden sind die beiden Funktionen dargestellt.

```
package biss.wise.BML;

import biss.BML.Evaluator;
import java.util.List;

public class MyClass
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
          extends Evaluator
{
public String fullName ( String partnerAcc)
{
  Object partner = getItem( null, partnerAcc);

  String titel = getField( partner, "Titel");
  String vorname = getField( partner, "Vorname");
  String nachname = getField( partner, "Name1");
  String anrede = getField( partner, "Anrede");

  return join( " ", anrede, titel, vorname, nachname);
}

public String fullName ( String geschlecht, String titel,
                            String vorname, String nachname)
{
  return join( " ", ("weiblich".equals( geschlecht) ? "Frau" : "Herr"),
                  titel, vorname, nachname);
}

public String summeLeben ( String vertraegeAcc)
{
  Object vertraege = getItem( null, vertraegeAcc);

  List bos = getChilds( vertraege, "BusinessObject");
  if (bos == null)
    return "";

  int n = bos.size();
  double sum = 0;
  for (int i = 0; i <bw: n; i++) {
    Object bo = bos.get( i);
    String code = (String)getItem( bo, "Produkt.Sparte.Code",
                           true);
    if (!"23".equals( code)) // LV
      continue;

    String bjb = (String)getItem( bo,
                           "BOVersion.Bruttojahresbeitrag", true);
    try {
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
        sum += Double.parseDouble( bjb);
      }
      catch (NumberFormatException ex) {
      }
   }

   return Double.toString( sum);
}

public String summeTotal ( String vertraegeAcc)
{
   Object vertraege = getItem( null, vertraegeAcc);

   List bos = getChilds( vertraege, "BusinessObject");
   if (bos == null)
      return "";

   int n = bos.size();
   double sum = 0;
   for (int i = 0; i <bw: n; i++) {
      Object bo = bos.get( i);
      String bjb = (String)getItem( bo,
                              "BOVersion.Bruttojahresbeitrag", true);
      try {
         sum += Double.parseDouble( bjb);
      }
      catch (NumberFormatException ex) {
      }
   }

   return Double.toString( sum);
}
}
```

Die Klasse "`MyClass`" ist von "`Evaluator`" abgeleitet und befindet sich im Paket "`biss.wise.BML`". Die Funktionen sind "`public`" und geben jeweils einen String zurück (alle Rückgabewerte und Argumente in BML sind Strings).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Kapitel 7

# BML und JavaScript

Bei der Erstellung von BML-Templatedateien muss zwischen drei Arten von JavaScript-Verwendungen unterschieden werden:

- Solche, die während der Transformationszeit ausgeführt werden sollen (diese sind innerhalb eines `<bw:script>...</bw:script>`-Blocks zu definieren,
- Solche, die zum Anzeigezeitpunkt der HTML-Seite ausgeführt werden können (diese sind innerhalb des HTML-typischen `<script>...</script>`-Blocks zu definieren und
- Solche, die auf Seiten des Clients ausschließlich in dessen Kontext ausgeführt werden sollen (diese sind innerhalb eines `<bw:cjscript>...</bw:cjscript>`-Blocks zu definieren); siehe hierzu auch den Abschnitt 9.13.

Zur Veranschaulichung dieser Unterscheidung dient die folgende Abbildung, die nochmals die zeitliche (chronologische) Reihenfolge der Schritte bei der Ausführung (Transformation) einer BML-Templatedatei darstellt (wir nehmen dabei an, dass der *B-Wise* Client veranlasst wurde, eine entsprechende Display-Seite anzuzeigen, welche mit Hilfe einer BML-Templatedatei definiert ist und dass das *B-Wise* Objektmodell - DOM - bereits im Speicher verfügbar ist).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

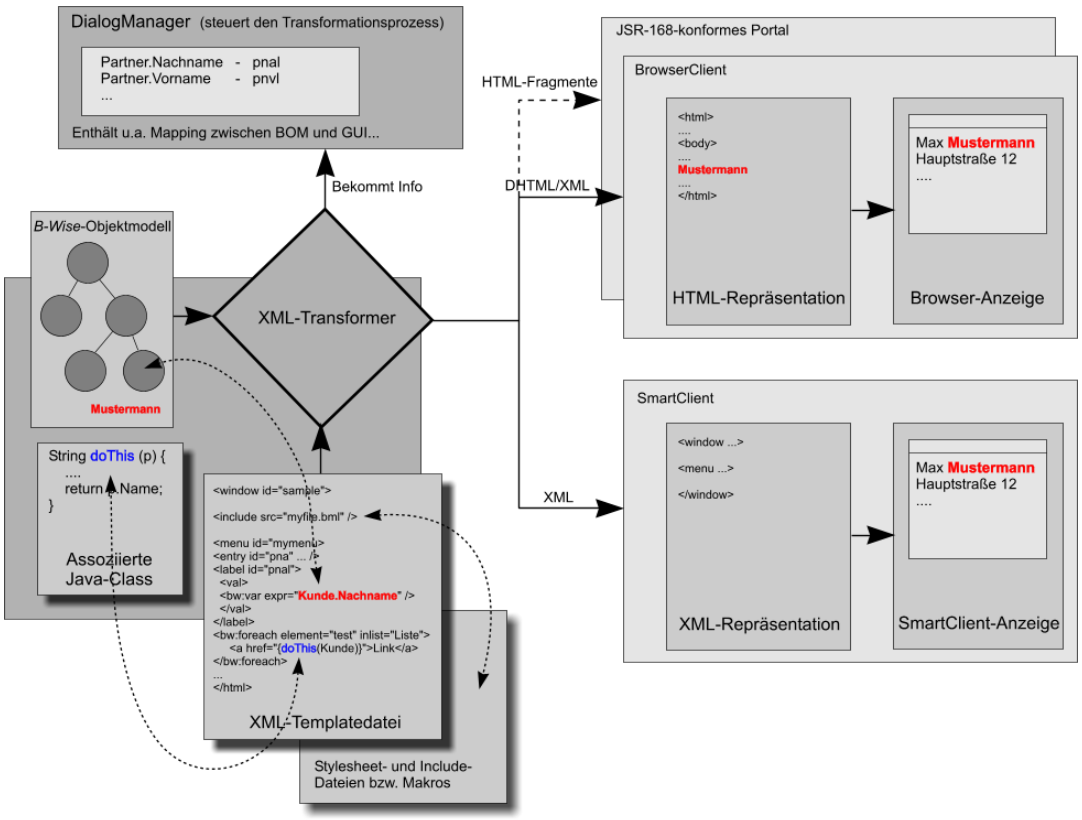



**Abb. 14: Schematischer Ablauf bei der Transformation einer BML-Seite**

1. Einlesen und Parsen der BML-Templatedatei
2. Transformieren der BML-Templatedatei (dabei wird der BML-Code "ausgeführt", d.h. die BML-Tags und deren Attribute werden evaluiert; u.a. werden `<bw:ig>`-Blöcke ignoriert, Bedingungen in `<bw:if>`-Tags ausgwertet und `<bw:var>`-Variablen durch die entsprechenden Werte des DOMs ersetzt; dies ist ein unidirektionaler Vorgang ! Innerhalb dieser Schrittes werden auch ggf. innerhalb eines `<bw:script>...</bw:script>`-Blocks definierte JavaScript-Funktionen ausgeführt; diese haben vollen Zugriff auf das aktuelle DOM des Servers)
3. Nach der Transformation wird der entsprechende Kontext verlassen (dies gilt auch für etwaige Variablen, die im Rahmen der Ausführung des Java-Codes entstanden sind)
4. Die Transformationsausgabe (je nach Art der BML-Seite BML-Code oder XHTML-Code ohne BML-Tags) wird an den Client übergeben; darin eingebettete `<bw:cjscript>`-Blöcke (im Falle einer reinen BML-Seite) bzw. `<script>`-Blöcke im Falle einer XHTML-Seite werden unangetastet gelassen
5. Parsen und Anzeigen des BML- bzw. XHTML-Codes (inkl. der Ausführung von Client-seitigem JavaScript-Code)


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Das folgende Beispiel zeigt eine solche Server-seitige JavaScript-Funktion:

```
<bw:script language="JavaScript" type="text/javascript">
<![CDATA[
function summeTotal ( vertraegeAcc ) {
  var ev = bsf.lookupBean( "bissBML");
  var vertraege = ev.getItem( null, vertraegeAcc);
  var bos = ev.getChilds( vertraege, "BusinessObject");
  if ( bos == null )
    return "";

  var n = bos.size();
  var sum = 0;
  for ( i = 0; i < n; i++ ) {
    var bo = bos.get( i);
    var bjb = ev.getItem( bo, "BOVersion.Bruttojahresbeitrag", true);
    var num = parseFloat( bjb);
    if ( !isNaN( num) ) {
      sum += num;
    }
  }

  return sum.toString();
}
]]>
</bw:script>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Kapitel 8

# BML-Regeln (Rules)

## 8.1 Einführung

Regeln bieten die Möglichkeit, definierte Elemente bzw. Teilebäume einer BML-Quelldatei in andere Elemente bzw. XML-Teilbäume zu transformieren. Sie bieten sich an, wenn wiederkehrende, individuelle Inhalte abgebildet werden sollen. Am besten lässt sich dies mit einem Beispiel veranschaulichen.

Nehmen wir an, wir möchten alle Dialogseiten einer Anwendung mit einer Überschrift (einer Titel-Zeile) versehen, die wie folgt aussehen soll (siehe Abb. 15).

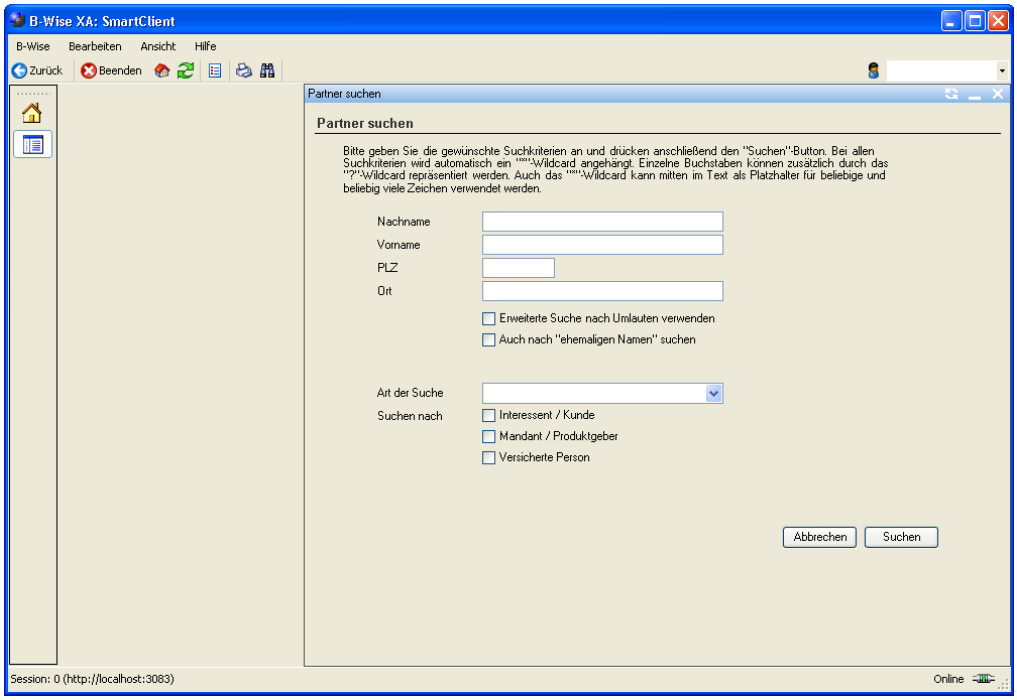



**Abb. 15: Überarbeiterer Beispieldialog mit Überschriftenzeile**

Wir verwenden hier den bereits im Tutorial (Abschnitt 2) definierten Dialog mit leichten Änderungen bzw. Erweiterungen. Der zugehörige BML-Code (gezeigt wird lediglich der Code der Überschrift) sieht wie folgt aus (Änderungen gegenüber dem in Abschnitt 2 definierten Dialog sind fett hervorgehoben):


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:label id="LA_Title" class="h1">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="10" eoy="10" ecx="230" ecy="30" />
  <bw:content><bw:t>Partner suchen</bw:t></bw:content>
</bw:label>
<bw:line id="DL_Title">
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
          eox="10" eoy="30" ecx="-10" ecy="30"/>
</bw:line>
```

Neben sprechenden IDs für den Titeltext wurde dieser mit einer Style-Class (einer Klasse für einen Layout-Stil) versehen und es wurde eine Linie unterhalb der Überschrift hinzugefügt.

Möchte man einen solchen Titel auf jeder Dialog-Seite wiederholen, gibt es mehrere Wege, dies zu erreichen:

### 1. Kopie

Man kann den zugehörigen Code kopieren, den Text der Überschrift ändern und in jeder Dialogseite einfügen; dies ist einfach und bei wenigen Seiten (weniger als 5) auch durchaus legitim. Nachteil dieser Lösung ist jedoch, dass Änderungen (z.B. die Vergrößerung des Abstands zwischen Titel-Text und Titel-Linie) an vielen Stellen durchgeführt werden müssen. Bei einer Vielzahl von Seiten (eine typische *B-Wise*-Anwendung hat ca. 250 - 500 Dialogseiten) ist bereits diese einfache Änderung kaum wirtschaftlich. Bei komplexen Teildialogen sind solche Änderungen praktisch nicht durchführbar.

### 2. Aggregation („Include“)

Eine wesentlich elegantere Lösung ist das Auslagern des Titel-Codes in eine separate Datei (in *B-Wise* „Include-Datei“ genannt und daher mit der Dateiendung „.inc“ versehen) und das Einfügen dieser Datei in jeden Dialog mit Hilfe des `<bw:include>`-Statements.

Dies löst zwar unser Maintanance-Problem (jede Änderung am Titel muss nur an einer einzigen Stelle durchgeführt werden), stellt und jedoch vor ein Neues: wie ändern wir den Titel in Abhängigkeit des verwendenden Dialogs ? Eine (durchaus legitime) Möglichkeit ist die Verwendung von Variablen bzw. sog. „Call-Parametern“.

Mit Hilfe von Variablen sieht die Lösung wie folgt aus:

```
<bw:var name="title_text" expr="Partner suchen" />
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:include src="System_DlgTitle.inc" />
```

Mit Hilfe von Call-Parametern sieht die Lösung recht ähnlich aus:

```
<bw:include src="System_DlgTitle.inc">
  <bw:param name="title_text" expr="Partner suchen" />
</bw:include>
```

In beiden Fällen haben wir den Code des Dialog-Titels in die Datei `System_DlgTitle.inc` ausgelagert und darin wie folgt modifiziert (die Änderung gegenüber dem oben dargestellten Code ist fett gedruckt):

```
<bw:label id="LA_Title" class="h1">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="10" eoy="10" ecx="230" ecy="30" />
  <bw:content><bw:var expr="$title_text" /></bw:content>
</bw:label>
<bw:line id="DL_Title">
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
          eox="10" eoy="30" ecx="-10" ecy="30"/>
</bw:line>
```

Anstelle des fixen Textes "Partner suchen" verwenden wir nun die Variable `title_text`, die im „aufrufenden" (besser: inkludierenden) BML-Code definiert wird. Der Unterschied zwischen den beiden o.g. Varianten liegt lediglich im Scoping, d.h. in der Gültigkeit der Variablen `title_text`. Während diese im ersten Beispiel ab ihrer Definition bis zur Redefinition oder bis zum Ende des Codes gültig ist, beschränkt sich ihre Gültigkeit im zweiten Beispiel auf den durch das `<bw:include>`-Element definierten Code-Block. Nach dem schließenden `</bw:include>` ist der Wert der Variablen `title_text` im zweiten Beispiel unbestimmt.

Für wenige variable Bestandteile ist diese Lösung durchaus akzeptabel. Bei vielen Variablen wird sie jedoch schnell unübersichtlich und auch „geschwätzig". Optimal (und für bestimmte Fälle die einzig mögliche Lösung) sind Regeln (BML-Rules).

### 3. Regeln

Die Lösung unseres Problems mit Hilfe von Regeln sieht wie folg aus:

Zunächst definieren wir ein neues Element, das wir `<bw:dlgtitle>` nennen (der Name darf nicht mit bereits innerhalb von BML definierten Element-Namen kollidieren und sollte


© Copyright BISS GmbH, 2001-2012 All Rights reserved

die Bedeutung des Elements möglichst gut wiederspiegel, ohne zu lang zu sein !). Anschließend verwenden wir in unserem Dialog (und in allen anderen Dialogen) dieses neue Element wie folgt:

```
<bw:dlgtitle text="Partner suchen" />
```

Wie man unschwer erkennen kann, ist diese Lösung nicht nur kürzer als alle anderen, sondern vor allem verständlicher und quasi „selbst-dokumentierend". Der Elementname gibt klar Auskunft über dessen Semantik und der Name des Attributs ist ebenfalls selbsterklärend.

Es stellt sich nur noch die Frage, wie aus dem o.g. Element `<bw:dlgtitle>` mit dem Attribut `text` der gewünschte Code wird ? Hier kommen endlich die BML-Rules zum Einsatz. Um das Element in den gewünschten Code zu transformieren, schreiben wir folgende Regel und speichern diese in einer Datei namens „Test_SampleRules.bwr" im selben Verzeichnis wie unsere Dialogdatei (die Datei erweiterung „.bwr" ist eine *B-Wise*-Konvention; letztlich sind sowohl der Dateiname als auch die Dateierweiterung gleichgültig).

```
<bw:rule element="bw:dlgtitle">
  <bw:label id="LA_Title" class="h1">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="10" eoy="10" ecx="230" ecy="30" />
    <bw:content><bw:var expr="@text" /></bw:content>
  </bw:label>
  <bw:line id="DL_Title">
    <bw:pos aox="0" aoy="0" acx="1000" acy="0"
           eox="10" eoy="30" ecx="-10" ecy="30"/>
  </bw:line>
</bw:rule>
```

Die Regel besteht im Wesentlichen aus den fett markierten Teilen:

- dem Element `<bw:rule>`, dessen `element`-Attribut den Namen unseres neuen Elements enthält sowie
- der Variablen `<bw:var>`, die das Attribut `text` unseres neuen Elements enthält und somit an der gewünschten Stelle ausgibt.

Der „Rest" ist unser bekannter Title-Code.

Jetzt müssen wir nur noch zu Beginn unserer Dialogdatei diese „Rule-Datei" mit Hilfe eines Include-Statements importieren:

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:include src="Test_SampleRules.bwr" />
```

*B-Wise* geht bei der Auswertung wie folgt vor:

Trifft der Transformer auf das Element `<bw:dlgtitle>` sucht es in den ihm bekannten Rules nach einem „Match"; findet der Transformer diesen Match, wir die Regel „getriggert" und der darin enthaltene Code ausgeführt (transformiert). Im vorliegenden Fall wird der in der Regel enthaltene BML-Code an die Stelle geschrieben, an der ursprünglich unser `<bw:dlgtitle>`-Element Stand. Dabei wird der Wert des Attributs `text` an die gewünschte Stelle ausgegeben.

Es ist leicht einzusehen, dass eine Erweiterung um weitere, variable Anteile übersichtlicher ist, als im Beispiel 2 (Include mit Variablen oder Parametern). So könnten wir die Position des Textes als weitere Parameter übergeben oder die Darstellung eines Bildes (Icons) in der Titelzeile erwägen.

Der von einer Regel emittierte Code ist nicht auf BML beschränkt sondern kann ebenfalls XHTML-Anteile enthalten, sofern die Darstellung innerhalb eines Browser-Controls gewünscht wird.

## 8.2 <bw:copy>

Das `<bw:copy>`-Element wird im Zuge der Regelverarbeitung benötigt, wenn aus dem Quell-Code Teile des Quell-Elements in den Ziel-Code übernommen werden sollen (Beispiele siehe unten).

**Syntax**

```
<bw:copy element="name" dropRoot="bool_value" exclude="element_list" />
```

| Attribut | Bedeutung |
|---|---|
| `element` | Name des Elements, das aus dem Quellcode kopiert werden soll; eine spezielle Bedeutung hat die Funktion „`root()`"; sie definiert das Root-Element, also das Regel-auslösende Element selbst (siehe Beispiele weiter unten); das Element wird zusammen mit all seinen Kind-Elementen kopiert, es sei denn, diese sind Bestandteil der „`exclude`"-Liste. Elemente eines Baumes werden durch einen Punkt getrennt, also z.B. „`root().h1`". |
| `dropRoot` | Gibt an, ob bei Angabe von „root()" für `element` (siehe oben), |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | auch das Element selbst (`dropRoot="false"`) oder nur dessen Inhalt (`dropRoot="true"`) kopiert werden sollen. |
| `exclude` | Gibt eine Komma-separierte Liste von Elementnamen an, die beim Kopieren des Quell-Baumes nicht mitkopiert werden sollen: Beispiel: `exclude="bw:v,bw:l"`. |

**Beispiel (im Rahmen einer Regel)**

```
<bw:rule element="bw:row">
   <bw:if expr="not( isEmpty( root()))">
      <bw:if expr="textContent( root().bw:v) ne ''">
         <div class="{concat('row ', root().@cls)}">
            <bw:copy element="root()" dropRoot="true" exclude="bw:v,bw:l" />
            <bw:var name="elemcnt"
                    expr="100/(count(root().*bw:l)+count(root().*bw:v))-1" />
            <bw:foreach element="root().*bw:l">
               <span class="{concat('l ', .@cls)}"
                           style="{concat( 'width:',$elemcnt,'%')}">
                  <bw:copy element="current()" dropRoot="true" />
               </span>
            </bw:foreach>
            <bw:foreach element="root().*bw:v">
               <span class="{concat('v ', .@cls)}"
                           style="{concat( 'width:',$elemcnt,'%')}">
                  <bw:copy element="current()" dropRoot="true" />
               </span>
            </bw:foreach>
         </div>
      </bw:if>
   </bw:if>
   <bw:if expr="isEmpty( root())">
      <div class="row"> </div>
   </bw:if>
</bw:rule>
```

Diese Regel dient der Erzeugung einer XHTML-Anzeige aus BML/XML-Code. Der zugehörige Quellcode (Fragment einer Vertragsanzeige) sieht wie folgt aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bw:col pos="left">
   <h1>Allgemeine Vertragsdaten</h1>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:row>
    <bw:l>Gesellschaft</bw:l>
    <bw:v><bw:var expr="Vertrag.Gesellschaftsname"/></bw:v>
  </bw:row>
  <bw:row>
    <bw:l>Vers.-Nummer</bw:l>
    <bw:v><bw:var expr="Vertrag.Versicherungsscheinnummer"/></bw:v>
  </bw:row>
  <bw:row>
    <bw:l>Vertragsstatus</bw:l>
    <bw:v><bw:var expr="Vertrag.Vertragsstatus.Value"/></bw:v>
  </bw:row>
</bw:col>
```

Die `<bw:row>` ist darin fett hervorgehoben. Die Regel transformiert den Quell-Code in das folgende Zielformat:

```
<div class="column left">
  <div class="title">
    <h1>Allgemeine Vertragsdaten</h1>
  </div>
  <div class="row">
    <span class="label">Gesellschaft</span>
    <span class="variable">Pfefferminzia AG</span>
  </div>
  <div class="row">
    <span class="label">Vers.-Nummer</span>
    <span class="variable right">244-M-877665.2</span>
  </div>
  <div class="row">
    <span class="label">Vertragsstatus</span>
    <span class="variable right">Lebend</span>
  </div>
</div>
```

und sorgt u.a. für eine Inhalts-abhängige Formatierung der einzelnen div-Elemente mit Hilfe von Stylesheets (CSS).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Kapitel 9

# BML Dialogelemente

## 9.1 Übersicht

BML wurde primär als abstrakte Dialogbeschreibungssprache entwickelt. Somit können in BML GUI-Dialoge in einer zielsystem*neutralen* Weise beschrieben werden. Die Transformation erzeugt zur Laufzeit aus dieser Beschreibung das zielsystem*spezifische* Format (z.B. XHTML), welches anschließend zur Darstellung der Dialoge innerhalb des jeweiligen Clients verwendet wird.

Folgende Elemente bzw. Controls stehen in der aktuellen BML-Version (BML 4.03) zur Verfügung:

- Portal (Basis-Fläche zur Darstellung aller Portlets und Fenster)
- Portlet (Fenster innerhalb eines Portals)
- Browser (Element zur Darstellung von mime-type-typisierten Inhalten, z.B. HTML-Seiten)
- Tree (Baumartige Darstellung hierarchischer Inhalte)
- List (spaltenweise Darstellung unterschiedlicher Inhalte - Texte und Images)
- DataGrid (tabellenartige Darstellung unterschiedlicher Informationen, insbesondere auch mit integrierten Widgets wie z.B. Checkbox, Entry oder Choice, Texten und Images)
- Choice (ComboBox)
- Editor (Element zur Erfassung und einfachen Bearbeitung von unformatiertem Text)
- TextEditor (Element zur Erfassung und Bearbeitung formatierter Texte)
- SyntaxEditor (Element zur Erfassung und Bearbeitung von Quellcode- bzw. Programm-Text)
- TextField (Entry, Eingabefeld für einzeilige Textinhalte)
- DateTime (Eingabefeld zur Erfassung von Datumswerten mit der Möglichkeit, Datumswerte aus einem PopUp-Kalender auszuwählen)
- Checkbox
- Button (Text- und/oder Image-Button)
- Radiobutton
- Compound (scrollbarer Dialogbereich, der u.a. auch eine logische Zusammenfassung von Controls ermöglicht)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

- Notebook (einzelne Panels, zwischen denen mit Hilfe von Tabs umgeschaltet werden kann, manchmal auch Notebook genannt)
- Menu (PullDown oder PopUp)
- BubbleHelp (Tooltip)
- Label (ein- oder mehrzeiliger Text mit einfachen Formatiermöglichkeiten)
- Toolbar
- PictureBox
- Deco-Elemente (Line, Rectangle, GroupBox, Circle, Image)

Viele der genannten Elemente können sowohl im Browser als auch im SmartClient verwendet werden, jedoch kann die Funktionalität in den Clients unterschiedlich sein. Der Grund hierfür liegt insbesondere in der Umsetzung der komplexen Elemente innerhalb des Browsers; um hier eine akzeptable Performance auch auf schächeren Systemen zu gewährleisten, muss auf Teile der Elemente und bei einigen Elementen auf Teile der Funktionalität verzichtet werden. Mit dem Aufkommen und der stärkeren Verbreitung leistungsfähiger Client-Systeme kann eine sukzessive Implementierung auch der komplexen Elemente (z.B. des DataGrid) für den BrowserClient erfolgen. Da der SmartClient .net kostenlos zur Verfügung gestellt wird, bedeutet dies jedoch keinen echten Nachteil.

Die folgende Tabelle zeigt, welche Elemente für welchen Client verfügbar sind.

| Element | SmartClient | BrowserClient |
|---|---|---|
| Portal | ✓ | ✓ |
| Portlet | ✓ | ✓ |
| Browser | ✓ | ✓ |
| Tree | ✓ | ✓ |
| List | ✓ | Eingeschränkt |
| DataGrid | ✓ | Nicht verfügbar |
| Choice | ✓ | ✓ |
| Editor | ✓ | ✓ |
| TextEditor | ✓ | Eingeschränkt |
| SyntaxEditor | ✓ | Eingeschränkt |
| TextField | ✓ | ✓ |
| DateTime | ✓ | Nicht verfügbar |
| Checkbox | ✓ | ✓ |
| Button | ✓ | ✓ |
| Radiobutton | ✓ | ✓ |
| Compound | ✓ | ✓ |
| Notebook | ✓ | Eingeschränkt |
| Menu | ✓ | ✓ |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | | |
|---|---|---|
| BubbleHelp | ✓ | ✓ |
| Label | ✓ | ✓ |
| Toolbar | ✓ | ✓ |
| PictureBox | ✓ | ✓ |
| Deco-Elemente | ✓ | ✓ |

Wie man unschwer erkennen kann, sind lediglich zwei Elemente (DataGrid und DateTime) im BrowserClient nicht verfügbar; einige der aufgeführten Elemente (z.B. der SyntaxEditor) sind funktional eingeschränkt, d.h. verfügen nicht über den vollen Funktionsumfang der Controls des SmartClients.

Für die Nutzung innerhalb des SmartClients stehen zusätzliche Elemente zur Verfügung, wie z.B. SyntaxEditor oder DesingerPane. Diese höherwertigen Elemente machen jedoch ausschließlich in spezialisierten Prozessen bzw. Kontexten Sinn und werden daher in dieser Dokumentation nicht weiter erläutert.

Die in der BML-Definition enthaltenen Widgets werden je nach Client (Browser oder Smart-Client) in den entsprechenden Ziel-Typ (DHTML-Control oder Smart-Control) transformiert und mit der maximal möglichen Funktionalität dargestellt. Dabei sorgt eine intelligente BML-Transformation für eine Browser-abhängige Optimierung. Damit wird ein Maximum an Funktionalität bei gleichzeitiger Gewährleistung der Offenheit und Kompatibilität erreicht. Für den *B-Wise* SmartClient werden zusätzliche Funktionalitäten bereitgestellt, die eine weitere Erhöhung des Benutzer-Komforts erlauben.

Die *B-Wise* Laufzeitumgebung stellt eine Bibliothek an DHTML-Controls für die Browser Internet Explorer ab Version 5.5 sowie Netscape ab Version 6 bzw. Mozilla ab Version 1.6 zur Verfügung. Diese Bibliothek ist erweiterbar und kann daher an die individuellen Bedürnisse des Unternehmens oder des jeweiligen Projekts angepasst werden.

BML definiert für die Verwendung innerhalb von Dialogbeschreibungen folgende Informationen:

- Metadaten (z.B. Autor oder Datum der Erstellung bzw. Änderung der BML-Datei)
- Dialogbeschreibung (Struktur, Inhalt und Anbindung der Controls an das Objekt-Modell der Business-Schicht)
- Layout-Informationen (Styles bzw. die Art und Weise der Darstellung wie z.B. Farben oder Fonts)
- Dynamik (Event-Handler und Key-Definitionen)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Zusätzlich ist eine Programmier-Schnittstelle (ein API) definiert, welches zur dynamischen (programmgesteuerten) Erzeugung von BML-Dialogen aus Prozessen heraus verwendet werden kann.

> **Hinweis:** Die API-Beschreibung auf den folgenden Seiten ist *B-Wise* intern und unterliegt laufenden Optimierungen. Sie ist nicht Bestandteil der BML-Spezifikation und ist für den Dialog-Designer kaum von Interesse; die API-Beschreibung ist in dieser Dokumentation lediglich als Leitfaden wiedergegeben, um dem Designer Hinweise auf mögliche dynamische Änderungen der Widgets durch die B-Wise Prozesse zu geben. Bitte beachten Sie, dass dieses API jederzeit und ohne Vorankündigung geändert werden kann !

Die o.g. BML Widgets können mit allen Elementen der BML (siehe auch Abschnitte 1 - 8) kombiniert werden. Dadurch können sehr mächtige Konstrukte inkl. hoch-dynamischer Dialogseiten erzeugt werden.

Die folgenden Abschnitte beschreiben die einzelnen Widgets/Controls im Detail. Neben einer Beschreibung der Widget-Funktionalität, der entsprechenden Syntax sowie der verfügbaren Parameter enthalten die Abschnitte auch typische Abbildungen der Widgets.

## 9.2 Metadaten

Folgende Metadaten-Tags sind verfügbar:

### 9.2.1 <bw:description>

Kennzeichnet einen deskriptiven Bereich, der normalerweise ausschließlich `<bw:meta>`-Tags enthält.

### 9.2.2 <bw:meta>

Das `<bw:meta>`-Tag definiert mit Hilfe der Attribute `name` und `val` ein beliebiges Name/Value-Pair. Aktuell existieren folgende (mit einer Semantik belegte) Werte:

| Element | Bedeutung |
|---|---|
| `author` | Autor der BML-Datei |
| `cdate` | Creation Date (Datum der Erstellung) |
| `mdate` | Modification Date (Datum der letzten Änderung) |
| `version` | Version der BML-Datei |
| `layer` | Name des Layers |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Beispiel:**

```
<bw:description>
  <bw:meta name="author" val="BISS GmbH" />
  <bw:meta name="version" val="1.3.44" />
  <bw:meta name="layer" val="Basisdialog" />
</bw:description>
```

# 9.3 Dialogbeschreibung

Die Dialogbeschreibung kann prinzipiell in zwei Teile gegliedert werden:

- Beschreibung des statischen Dialoganteils
- Beschreibung des dynamischen Dialoganteils

Der statische Dialoganteil enthält die Dialogelemente sowie ihre Positionen bzw. innerhalb der Styles definierten Layout-Anteile. Hierzu zählen z.B. Portlets, Labels, Eingabefelder, Listen, Trees, Deko-Elemente etc.

Der dynamische Dialoganteil enthält Informationen über die Verbindung der Dialogelemente mit dem Modell (bzw. den einzelnen Attributen) sowie Informationen über das Verhalten, das mit Hilfe bestimmter Elemente ausgelöst werden soll (also z.B. das Verhalten, das eine Auswahl eines Listenelements triggern soll).

Beide Teile sind in den folgenden Abschnitten beschrieben - gegliedert nach den einzelnen Widgets.

## 9.3.1 Allgemeine Attribute

In diesem Abschnitt sind diejenigen Attribute beschrieben, die jedes Element enthält (allgemeine, querschnittliche Attribute). In Klammern hinter dem Attribut-Namen ist der Typ des Attributs aufgeführt („B“ = Boolean, „S“ = String“, „E“ = Enumeration/Aufzählung, „I“ = Interger).

### 9.3.1.1 id (S)

Jedes Element (mit Ausnahme der Deco-Elemente) muss eine innerhalb des gesamten BML-Dialogs eindeutige ID besitzen. Diese ID ermöglich den Zugriff auf das Element seitens des Servers und dient damit primär der Kommunikation zwischen Dialog und Modell. Zum „gesamten“ Dialog zählen auch alle inkludierten Dialoganteile !

© Copyright BISS GmbH, 2001-2012 All Rights reserved

#### 9.3.1.2 enabled (B)

Wird dieses Attribut auf den Wert „false“ oder „0“ gesetzt, ist das entsprechende Element inaktiv, d.h. nimmt keine Eingaben entgegen und reagiert auch nicht auf Benutzerkommandos (z.B. Clicks). Zusätzlich wird das Element entsprechend visuell dargestellt.

#### 9.3.1.3 visible (B)

Wird dieses Attribut auf den Wert „false“ oder „0“ gesetzt, wird das entsprechende Element nicht dargestellt.

#### 9.3.1.4 state (E)

Das state-Attribut definiert einen logischen Zustand des entsprechenden Elements. Folgende Zustände und deren Bedeutungen sind aktuell definiert:

| Zustand | Beschreibung |
|---|---|
| `initial` | Definiert den Normalzustand, also: nicht required, nicht erroneous |
| `required` | Definiert ein MUSS-Element. Dieses Element wird gemäß der Definition von MUSS-Feldern dargestellt. |
| `erroneous` | Definiert ein Element mit ungültigem bzw. fehlerhaftem Inhalt; dieses Element wird gemäß der Definition für fehlerhafte Elemente dargestellt. |
| `disabled` | Definiert einen Zustand, in dem das Element inaktiv und unbedienbar ist; in vielen Fällen wird es insgesamt abgegraut dargestellt |
| `readonly` | Definiert einen Zustand, in dem ein Element für Datenerfassung keine Daten entgegen nimmt, sondern nur der Anzeige dient. |

#### 9.3.1.6 class (S)

Definiert den Namen der Darstellungs- (Style-) klasse.

#### 9.3.1.7 datasrc (S)

Das `datasrc`-Attribut ist eines der wichtigsten Attribute. Es verbindet ein Widget mit der entsprechenden Datenquelle innerhalb des Objektmodells. Diese Verbindung erfolgt über Namen. Im Normalfall handelt es sich um die Namen der *B-Wise* Properties (Daten-Attribute), also z.B. um „Partner.Nachname“.

Der *B-Wise* Dialogmanager sorgt im Normalfall dafür, dass sich jede Änderung der Datenquelle innerhalb der Business-Tier im Dialog widerspiegelt und dass bei jeder Änderung des Wertes der Datenquelle innerhalb des Dialogs (Client-Tier) der Server benachrichtigt wird und damit Gelegenheit erhält, die entsprechende Datenquelle zu aktualisieren.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

#### 9.3.1.8 tabindex (I)

Definiert den Tabindex (Ganze Zahl) und damit die Fokus-Reihenfolge beim Betätigen der TAB-Taste. Der Index muss nicht konsekutiv sein; die Tabreihenfolge wird lediglich durch einen Wertevergleich ermittelt, so dass ein Element mit einem tabindex="23" einem Element mit einem tabindex="12" folgt, sofern kein Element einen tabindex mit einem Wert zwischen 13 und 22 besitzt. Dies ist wichtig, wenn generische Includes verwendet werden sollen.

#### 9.3.1.8 keep (B)

Definiert, falls „false", dass das zugehörige Element bei der Transformation nicht in das Ziel-DOM übernommen wird, also effektiv aus dem Dialog entfernt wird. Default-Wert ist „true".

### 9.3.2 Allgemeine („innere") Tags

Neben allgemeinen Attributen können Elemente auch über allgemeine (innere, d.h. nur innerhalb eines Elements gültige) Tags verfügen. Diese Tags können einerseits unmittelbar als Child-Elemente des betreffenden Elements vorkommen oder mit Hilfe einer Reference-ID („refid"-Attribut innerhalb des Tags) an das Parent-Element gebunden werden. Im letzteren Fall können die Tags an einer beliebigen Stelle innerhalb des Dokuments (innerhalb des Root-Elements) vorkommen. Im folgenden sind zwei Beispiele für die Nutzung aufgeführt:

Direkte Nutzung innerhalb des Elements:

```
<bw:entry id="myentry">
  <bw:pos aox="0"  aoy="0"  acx="0"   acy="0"
           eox="20" eoy="10" ecx="220" ecy="30"
  />
  .. weitere Inhalte ..
</bw:entry>
```

Mit Hilfe von `refid` referenzierte Nutzung:

```
<bw:entry id="myentry">
  .. weitere Inhalte ..
</bw:entry>

<bw:pos refid="myentry"
  aox="0"  aoy="0"  acx="0"   acy="0"
  eox="20" eoy="10" ecx="220" ecy="30"
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
/>
```

Grundsätzlich lassen sich alle Elemente mit Hilfe der `refid` in ein Parent-Element „einhängen“.

Die querschnittlich verwendbaren Tags sind im folgenden beschrieben.

#### 9.3.2.1 <bw:style>

Das `<bw:style>`-Element enthält eine Layout-Beschreibung für das entsprechende Element. Damit ist eine direkte Style-Angabe möglich. Alternativ kann das `class`-Attribut verwendet werden oder eine regelbasierte Zuordnung zwischen Style und Element erfolgen. Hierzu müssen entsprechende Style-Templates definiert werden (siehe Abschnitt 9.9).

#### 9.3.2.2 <bw:pos>

Definiert die Position des Elements sowie sein Verhalten bei Größenänderungen des umgebenden Parents. Das `<bw:pos>`-Tag verfügt über folgende Attribute:

| Element | Beschreibung |
|---|---|
| `asmode (E)` | (Optional) Definiert, ob das Element seine Größe (Boundary) automatisch dem Inhalt anpasst; mögliche Werte sind:<br>„none“ = keine automatische Größenanpassung (Default-Wert)<br>„width“ = die Breite wird automatisch angepasst (Höhe ist fest definiert)<br>„height“ = die Höhe wird automatisch angepasst (Breite ist fest definiert)<br>Diese Angabe macht vor allem beim Label und Compound Sinn. |
| `policy (E)` | (Optional) Definiert, ob das Element beim Flow-Layout im Layout auftaucht, wenn es unsichtbar ist; mögliche Werte sind:<br>„always“ = Das Element taucht immer im Layout auf (d.h. es nimmt seinen definierten Platz ein, auch wenn es unsichtbar ist)<br>„visible“ = Das Element taucht nur im sichtbaren Zustand im Layout auf |
| `unit (E)` | (Optional) Definiert, welche Einheiten für die Positionsangaben verwendet werden sollen; mögliche Werte sind:<br>„pixel“ = Es werden Pixel verwendet (Default-Wert)<br>„sysfont“ = Es wird ein abstraktes Maß verwendet, welches Dialoge unabhängiger von der eingestellten Fontgröße macht (eine Alternative zu „sysfont“ ist die Dialogskalierung, die der SmartClient zur Verfügung stellt; diese skaliert auch Images und HTML-Inhalte) |
| `hmin (I)` | (Optional) Definiert die Mindesthöhe eines Elements; wird ein Element derart definiert, dass es bei einer Größenänderungs des Parents „schrumpft“, bestimmt dieser Wert die Mindesthöhe, die nicht unterschritten wird |
| `wmin (I)` | (Optional) Definiert die Mindestbreite eines Elements; wird ein Element derart defi- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | niert, dass es bei einer Größenänderungs des Parents „schrumpft“, bestimmt dieser Wert die Mindestbreite, die nicht unterschritten wird |
| `rel` | Definiert eine relative Positionierung des Elements zum unmittelbar vorhergehenden Element. Das `rel`-Attribut kann die Werte „`xflow`“ oder „`yflow`“ annehmen; „`xflow`“ bestimmt, dass eine relative Posionierung in X-Richtung, „`yflow`“ eine relative Positionierung in Y-Richtung stattfinden soll. Wird `rel` verwendet, dann bestimmen die Attribute `eox` bzw. `eoy` die relative Position der linken oberen Ecke (in Pixel) und die Attribute `ecx` und `ecy` die relative Position (in Pixel) der rechten unteren Ecke zum vorhergehenden Element. |
| `aox` | X-Wert (Abstand vom linken Rand des Parents) des Origin- bzw. Anchor-Punktes (links-oben) angegeben in Promille (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). „aox“ steht für „anchor origin x“. |
| `aoy` | Y-Wert (Abstand vom oberen Rand des Parents) des Origin- bzw. Anchor-Punktes angegeben in Promille (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). |
| `acx` | X-Wert (Abstand vom linken Rand des Parents) des Corner-Punktes (rechts unten) angegeben in Promille (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). „acx“ steht für „anchor corner x“. |
| `acy` | Y-Wert (Abstand vom oberen Rand des Parents) des Corner-Punktes (rechts unten) angegeben in Promille (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). |
| `eox` | X-Wert (absolute Position angegeben in Pixel oder SysFont) des linken oberen Punktes des Elements in Bezug auf den Anchor-Origin (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). „eox“ steht für „extension origin x“. |
| `eoy` | Y-Wert (absolute Position angegeben in Pixel oder SysFont) des linken oberen Punktes des Elements in Bezug auf den Anchor-Origin (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). |
| `ecx` | X-Wert (absolute Position angegeben in Pixel oder SysFont) des rechten unteren Punktes des Elements in Bezug auf den Corner-Anchor (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). |
| `ecy` | Y-Wert (absolute Position angegeben in Pixel) des rechten unteren Punktes des Elements in Bezug auf den Corner-Anchor (siehe auch Beschreibung des *B-Wise*-Layouters im Abschnitt 9.11). |

Bezüglich der Funktionsweise des *B-Wise* Layouters siehe auch Abschnitt 9.11.

#### 9.3.2.3 <bw:keys>

Das `<bw:keys>`-Tag ermöglicht die Definition von Tastatur-Zuordnungen zu bestimmten Befehlen. Damit können sowohl Events getriggert als auch Befehle auf dem Server mit Hilfe der Tastatur ausgelöst werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Das <bw:keys>-Tag definiert sowohl die Referenz auf eine allgemeingültige Datei, welche entsprechende Zuordnungen enthält als auch den Rahmen für die einzelnen Zuordnungen eines Widgets.

Das <bw:keys>-Tag kann ein Innertag (<bw:include>) haben. Dieses definiert den Pfadnamen einer Datei, welche die in BML definierten Key-Zuordnungen enthält. Damit können querschnittlich verwendbare Templates für Tasten-Zuordnungen definiert werden

Das <bw:keys>-Tag kann eine beliebige (0...n) Anzahl <bw:key>-Tags enthalten. Ein <bw:key>-Tag assoziiert eine Taste mit einem Event bzw. einem Befehl. Hierfür besitzt das <bw:key>-Tag zwei Attribute:

| Attribut | Beschreibung |
|---|---|
| name | Bezeichnung der Taste; es wird immer ein Prefix verwendet, folgende Prefixe stehen zur Verfügung:<br><br>C „Ctrl“-Modifier<br>A „Alt“-Modifier<br>S „Shift”-Modifier<br><br>Es sind beliebige Kombinationen der Modifier “A”, “C” und “S” erlaubt. |
| method | Befehl (auf dem Server) oder Event; Events haben einen „#“ vor dem Namen, z.B. „#click“. |

#### 9.3.2.4 <bw:events>

Das <bw:events>-Tag ermöglicht die Zuordnung von Events zu Befehlen (Commands bzw. Methods) auf dem Server.

Das <bw:events>-Tag kann eine beliebige (0...n) Anzahl <bw:event>-Tags enthalten. Ein <bw:event>-Tag assoziiert ein Event mit einem Befehl (Command) auf dem Server. Hierfür besitzt das <bw:event>-Tag zwei Attribute:

| Attribut | Beschreibung |
|---|---|
| name (S) | Bezeichnung des Events (siehe Abschnitt 9.3.3 bzw. Anhang G). |
| method (S) | Befehl (auf dem Server) der beim Auslösen des Events ausgeführt werden soll. |
| type (S) | (Optional) Definiert, ob es sich um eine Metode des Servers handelt (Default-Wert) oder um eine (Skript-) Methode des Clients (type=“cjscript“). |

Als Beispiel seien an dieser Stelle folgende Events genannt:


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**clicked**
Das `clicked`-Event wird ausgelöst, wenn ein Anwender auf ein Element mit Hilfe der linken Maustaste clickt. Es gehört zur Gruppe der „physischen" Events, also Events, die durch eine definierte Handlung (in diesem Falle den Mausclick) ausgelöst werden.

**focuslost**
Das `focuslost`-Event wird ausgelöst, wenn das Element den Fokus verliert (z.B. wenn der Benutzer mit Hilfe der Tab-Taste ein anderes Element fokussiert).

**validate**
Das `validate`-Event wird ausgelöst, wenn der Benutzer den Inhalt eines Widgets validiert. Dies ist abhängig vom Widget und kann darüberhinaus innerhalb einer Anwendung recht variabel festgelegt werden. `validate` gehört zur Gruppe der sog. „logischen" Events, d.h. Events, die eine variable Zuordnung zwischen Benutzeraktion und Event erlauben. Im vorliegenden Fall wird z.B. bei einem Entryfield das validate Event durch mehrere physische Events ausgelöst nämlich durch `focuslost` oder das Drücken der Enter-Taste.

`validate` kann sowohl mit Hilfe der Maus als auch mit Hilfe der Tastatur getriggert werden. So wird `validate` z.B. bei einem Button dann getriggert, wenn der Benutzer mit der linken Maustaste auf den Button clickt oder die Enter- oder Space-Taste drückt, wenn der Button fokussiert ist.

#### 9.3.2.5 <bw:tooltip>

Ein `tooltip`-Element stellt ein PopUp-Rechteck dar, das einem beliebigen Widget zugeordnet werden kann und erscheint, wenn der Benutzer eine definierte Zeit mit dem Mauszeiger über dem Widget verharrt.

### Syntax

```
<bw:tooltip
  id="widget-id"
  class="style-class"
  >
  <bw:tth>Das ist der Titel</bw:tth>
  <bw:ttb>Das ist der eigentliche Text der PopUp-Hilfe</bw:ttb>
</bw:tooltip>
```

### Parameter

| Parameter | Beschreibung |
|---|---|

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID einer `<bw:div>`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Diese ID wird nur benötigt, wenn das Tooltip außerhalb eines Widgets definiert und über eine Referenz angesprochen wird. Wird das Tooltip innerhalb eines Widgets definiert, kann die ID entfallen. Es ist dann allerdings nicht möglich, das Tooltip programmgesteuert anzusprechen. |
| `class (S)` | (Optional) Eine Referenz auf eine Style-Template-Definition, welche die Darstellung (das Layout) des Widgets bestimmt. Die unterstützten Regeln sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Tooltips verwendet werden. Angaben innerhalb des <bw:style>-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs. <bw:style>-Tags sind im <bw:menu>-Element selbst sowie in <bw:mi>-Elementen erlaubt. |

### API-Funktionen

**`setTooltip( title, text )`**

Setzt den Inhalt des Tooltips; dies sowohl für den Titel als auch für den Body-Text. Dies ist ein entsprechend formatierter Wert.

### Hinweise

Der Text selbst wird als Inhalt des `<bw:tooltip>`-Tags angegeben. Damit können auch `<bw:var>`-Tags als Inhalte sowie alle anderen gültigen BML-Konstrukte verwendet werden. Der Text kann gemäß den allgemeinen BML-Textregeln (siehe auch `<bw:editor>` oder `<bw:label>`) formatiert werden.

### Beispieldarstellung

```
<bw:tooltip id="mytip" class="mystyle">
  <bw:tth><bw:var expr="HelpDB[parent.id].Title" /></bw:tth>
  <bw:ttb><bw:var expr="HelpDB[parent.id].Text" /></bw:ttb>
</bw:tooltip>
```

## 9.3.3 Server-API

**Hinweis:** Die API-Beschreibung auf den folgenden Seiten ist *B-Wise* intern und unterliegt laufenden Optimierungen. Sie ist nicht Bestandteil der BML-Spezifikation und ist für den Dialog-Designer kaum von Interesse; die API-Beschreibung ist in dieser Dokumentation


© Copyright BISS GmbH, 2001-2012 All Rights reserved

lediglich als Leitfaden wiedergegeben, um dem Designer Hinweise auf mögliche dynamische Änderungen der Widgets durch die *B-Wise* Prozesse zu geben. Bitte beachten Sie, dass dieses API jederzeit und ohne Vorankündigung geändert werden kann !

Das Server-API definiert die Funktionalität der einzelnen Dialog-Elemente aus Sicht des *B-Wise* Servers. Diese API findet seine Entsprechung in der Funktionalität des Clients. Dieses API bedeutet jedoch nicht, dass der Server über einen Call Funktionen im Client synchron aufrufen kann. Die über das API definierten Funktionen werden ausschließlich im Zuge der Return-Werte eines Client-RPC-Calls aufgerufen.

Dieses Design entspricht dem von Meszaros 1995 definierten Half-Object-Pattern, in dem das UI-Object (z.B. ein Menü) aus einer Objekt-Hälfte im Client sowie einer zweiten Objekt-Hälfte auf dem Server besteht, die beide über ein Protokoll miteinander verbunden sind. Wichtig im vorliegenden Fall ist jedoch die Abwandlung, dass keine unmittelbare Call-Funktionalität vom Server aus verfügbar ist, da die Kommunikation zwischen Client und Server grundsätzlich unidirektional (Client-initiiert) erfolgt. Auf dem Server aufgerufene Client-Funktionen werden also erst dann zum Client übertragen, wenn dieser selbst eine Funktion im Server initiiert hat und diese Funktion zurückkehrt.

Das folgende Beispiel möge dies verdeutlichen.

Eine Funktion im Server möchte gerne ein Eingabefeld unsichtbar machen (d.h. das `visible`-Attribut auf false setzen). Die Funktion holt sich vom DialogManager die Referenz auf den aktuellen Dialog und das gewünschte Control. In beiden Fällen die eindeutige ID sowohl des Dialogs als auch des Controls der Referenzierung. Schließlich wird die entsprechende Funktion (`setVisible`) des Controls mit dem erforderlichen Parameter aufgerufen.

```
...
getDlgManager().getPortlet( "Test_SampleDialog").getControl(
"TF_Name").setVisible( false);
...
```

Nach Ausführung dieses Codes wird der eigentliche „Call“ (also Aufruf der `setVisible`-Funktion innerhalb des Controls) beim Returnwert des aktuellen XML-RPC-Call des Clients an diesen übertragen und entsprechend dispatched und ausgeführt.

Es ist daher sinnvoll, die XML-RPC-Calls des Clients als eine Art „Dialog-Transaktionen“ zu betrachten; dabei wird die Transaktion durch eine Aktion innerhalb des Clients (z.B. einen Mausclick oder einen Timer-gesteuerten Request) initiiert und endet mit dem Return des entsprechenden XML-RPC-Calls. Dieser Return beinhaltet gesammelt alle im Rahmen dieser

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Transaktion entstandenen Zustandsänderungen, Werteänderungen oder auch neue Elemente (z.B. MessageBoxen oder Dialoge); sie werden zum Client propagiert, wo sie ausgewertet und ausgeführt werden.

Die nachfolgende Abbildung stellt diese Transaktion schematisch dar.

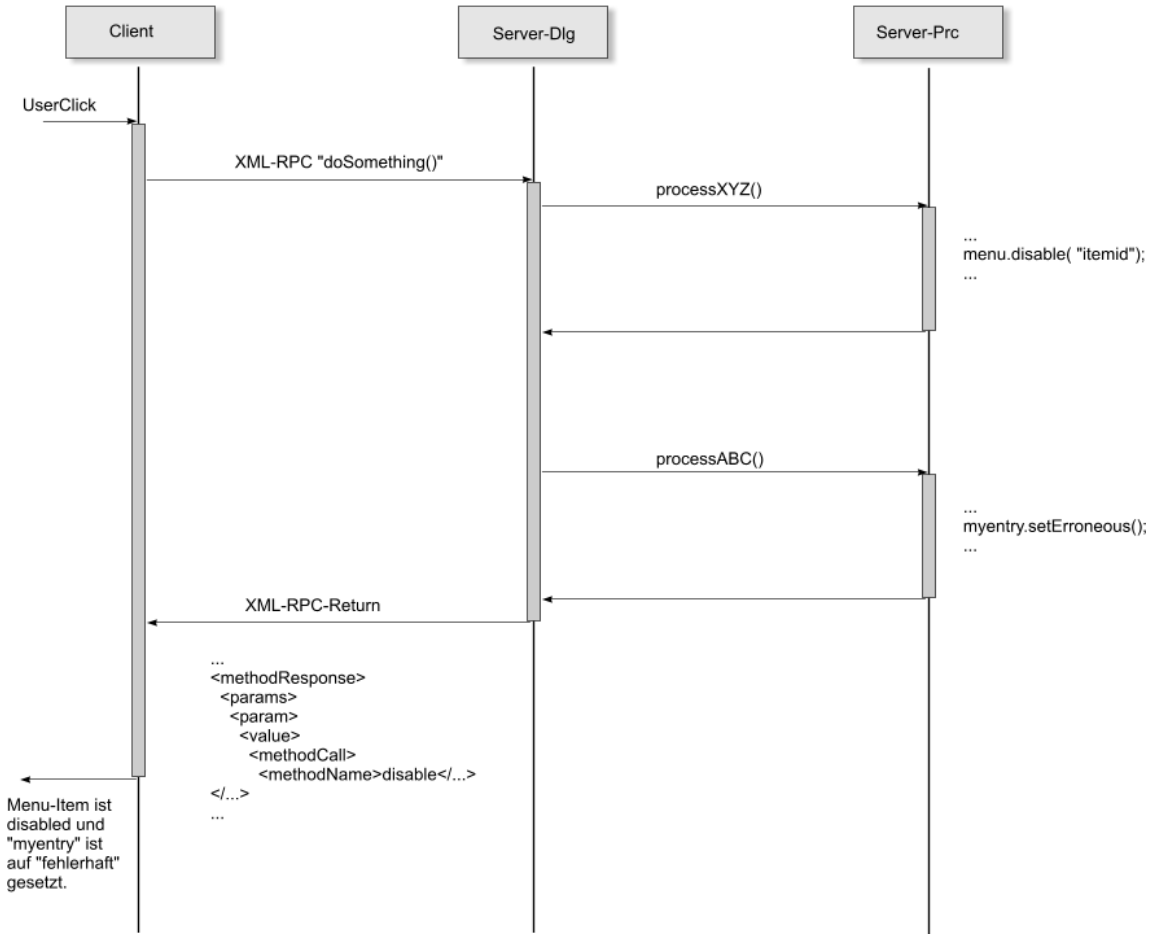



**Abb. 16: Transaktionsdiagramm zwischen Client und Server**

Es ist möglich, dem Return ein „Continue"-Flag mitzugeben, welches dem Client signalisiert, dass dieser Returnwert lediglich einen Teil der eigentlichen Rückhabeinformation beinhaltet und der Client sich somit unmittelbar nach Verarbeitung dieses (ersten) Teils wieder an den Server wenden soll, damit dieser mit weiteren Teilen fortfahren kann.

Da diese Calls normalerweise asynchron stattfinden, kann auf diese Weise z.B. eine Fortschrittsinformation signalisiert werden, ohne dass der Benutzer in seinen Aktionen behindert wird.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Im folgenden sind einige Beispiele für die Nutzung des APIs sowie dessen Abbildung auf den Rückgabewert eines XML-RPC-Calls wiedergegeben.

```
<bw:methodResponse>
  <bw:params>
    <bw:param>
      <bw:value>
        <bw:![CDATA[

        ... ResponseCalls (siehe unten) ...

        ]]>
      </bw:value>
    </bw:param>
  </bw:params>
</bw:methodResponse>

<!-- Beispiele für ResponseCalls -->
<bw:cmd id="myentry" type="entry" name="setVal">
  <bw:val>Mustermann</bw:val>
</bw:cmd>

<bw:cmd id="mylist" type="list" name="insert">
  <bw:index>1</bw:index>
  <bw:state>selected</bw:state>
  <bw:data>
    <bw:lbody>
      <bw:li id="0">
        <bw:lc class="PLZ">
          <bw:img src="images/Partner02_bl.gif"/>
          26689
        </bw:lc>
        <bw:lc>
          Mustermann
        </bw:lc>
      </bw:li>
      <bw:li id="1">
        <bw:lc class="PLZ">
          <bw:img src="images/Partner02_ge.gif"/>
          65436
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
          </bw:lc>
          <bw:lc>
            Petersen
          </bw:lc>
        </bw:li>
      </bw:lbody>
    </bw:data>
</bw:cmd>

<!-- Beispiel für BrowserClient (hier nach Transformation der
     param-Section) -->
<bw:cmd name="create">
  <bw:mbox>
    <bw:div id="mymessagebox">
      <bw:h1>Titel der Box</bw:h1>
      <bw:p>Dies ist meine tolle Messagebox.</bw:p>
      <bw:div class="button">
        ... weitere Tags für den DHTML-Button ...
      </bw:div>
    </bw:div>
  </bw:mbox>
</bw:cmd>

<!-- Beispiel für SmartClient -->
<bw:cmd name="create">
  <bw:mbox>
    <bw:mbox id="mymessagebox" mbtype="mbok">
      <bw:title>Titel der Box</bw:title>
      <bw:val>Dies ist meine tolle Messagebox.</bw:val>
    </bw:mbox>
  <bw:mbox>
</bw:cmd>

<!-- Beispiel für das Löschen eines Portlets -->
<bw:cmd id="pc" type="portletcontainer" name="delete">
  <bw:id>myportlet</bw:id>
</bw:cmd>
```

**Hinweise**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

- Das `id`-Attribut innerhalb des `<bw:cmd>`-Tags entspricht der Target-ID des Widgets, das `name`-Attribut enthält den Namen des API-Calls; das `type`-Attribut enthält den Typ („Klasse“) des Widgets und ermöglicht dabei entsprechende Typprüfungen.
- Wird keine Target-ID angegeben, wird der Call "auf oberster Ebene", d.h. im Kontext des Clients ausgeführt (z.B. für die Anzeige modaler Dialoge); dies entspricht quasi dem Call auf das "Root"-Element des Dialogs
- Im Falle neuer Elemente (z.B. MessageBox oder auch Portlet) wird kein id-Attribut verwendet und als Tag der Typ des Elements angegeben (im obigen Beispiel `<bw:mbox>`); der Inhalt muss in jedem Falle durch die Transformation laufen.
- Befinden sich (nach der Transformation für den BrowserClient) `<bw:script>`-Tags innerhalb des neuen Elements (z.B. für die DHTML-Controls), werden diese im BrowserClient "gesammelt" und anschließend explizit vom JavaScript-Dispatcher ausgeführt (da der Browser über XML-HTTP übermittelte HTML-Fragmente zwar ins DOM einhängen, aber keine `<bw:script>`-Tags ausführen kann).
- Als XML-RCP-Response kommen somit ausschließlich "Calls" in Form von `<bw:cmd>`-Command-Tags; diese werden im Client (vor allem im SmartClient) nicht nur für Dialoge sondern auch für andere Komponenten (z.B. Office-Anbindung) verwendet.

## 9.4 Internationalisierung

Um BML-Seiten in unterschiedlichen Sprachen darstellen zu können ist es erforderlich, sie an das Internationalisierungs-Subsystem von *B-Wise* anzuschließen. Dies erfolgt mit Hilfe des `<bw:t>`-Tags bzw. der `t()`-Funktion.

Alle Texte, die in ein `<bw:t>`-Tag eingeschlossen sind, werden vom *B-Wise* Internationalisierungs-System ausgewertet und entsprechend übersetzt (siehe hierzu auch das Dokument „*B-Wise* Internationalisierung“). Dasselbe gilt für die t()-Funktion. Das folgende Beispie zeigt eine Verwendung sowohl des `<bw:t>`-Tags als auch der `t()`-Funktion innerhalb eines Portlets.

```
<bw:portlet
   xmlns:bw=http://www.biss-net.com/2001/bwise/ehtml
   id="portletid"
   title="{t('Text innerhalb der Titlebar')}"
   target="cont"
   state="normal"
   mode="normal">

  <bw:label id="labelid">
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
      eox="30" eoy="200" ecx="630" ecy="220" />
    <bw:content>
      <bw:t>Dies ist der Text des Labels, der übersetzt wird.</bw:t>
    </bw:content>
  </bw:label>
</bw:portlet>
```

Das *B-Wise* Internationalisierungs-Subsystem speichert alle Texte sowohl der Originalsprache als auch aller Übersetzungen innerhalb einer Datenbank. Dabei dient der Originaltext selbst als Index in die Datenbank.

In bestimmten Situationen kann es vorkommen, dass der Originaltext je nach Kontext unterschiedliche Übersetzungen hat (klassisches Beispiel: „Leiter"). Um dies zu ermöglichen, kann zusammen mit dem `<bw:t>`-Tag als auch der `t()`-Funktion eine id innerhalb des entsprechenden Begriffs übergeben werden, die zur Differenzierung der übersetzen Inhalte dient. Hierzu ebenfalls zwei Beispiele:

```
<bw:t>@1Leiter</bw:t>

<bw:t>@2Leiter</bw:t>
```

Als Übersetzung steht dann in der entsprechenden Internationalisierungs-Datenbank:

```
@1Leiter  <-> leader
@2Leiter  <-> ladder
```

*B-Wise* erlaubt als zusätzliche ID nach dem @-Zeichen lediglich ein einzelnes Zeichen, was jedoch in der allermeisten Fällen ausreichend sein dürfte, da dem Autor keine Begriffe bekannt sich, die in unterschiedlichen Sprachen mehr als 9 unterschiedliche Bedeutungen haben können und die alleine innerhalb der Anwendung stehen.

## 9.5 Text-Formate

Texte können innerhalb von BML mit Hilfe einfacher Regeln (Tags) formatiert werden. Werden komplexe Layouts verlangt, ist das `<bw:browser>`-Widget zu verwenden; dieses ermöglicht die Anzeige von XHTML-Texten (gemischt mit BML-Tags) mit allen in XHTML erlaubten Formaten.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Folgende Text-Formatregeln stehen unmittelbar (d.h. auch ohne Nutzung des `<bw:browser>`-Widgets) in den entsprechenden BML-Widgets zur Verfügung:

| Format-Tag | Beschreibung |
|---|---|
| `<b>` | Boldface (Fettdruck) |
| `<i>` | Italics (Kursiv) |
| `<u>` | Underline (Unterstrichen) |
| `<br />` oder `\n` | Line break; definiert einen expliziten Zeilenumbruch; ansonsten wird der Text gemäß den Breitenangaben formatiert. |
| `<link>` | Ein Link bzw. Command (der Text wird gemäß definierbarer Styleinformationen als Link formatiert). Das `<bw:link>`-Tag benötigt das `cmd`-Attribut, welches ein gültiges Command enthalten muss also z.B.<br><br>`<bw:link cmd="showHelp">Text</bw:c>`<br><br>Zusätzlich (und optional) kann das `<bw:link>`-Tag noch die Attribute `class` (gibt eine Style-Datei an) sowie `title` (definiert eine BubbleHelp für den Link) enthalten. |
| ~ | (Tilde) Stellt einen Sonderfall der Formatierung dar und dient primär der Definition einzelner unterstrichener Buchstaben, die zusätzlich automatisch mit der Bedeutung „setFocus“ in die Keys-Referenztabelle eingetragen werden (mit dem „Alt“-Modifier). Beispiel für einen Text einer Checkbox: „Auch ~Aquarien versichern“; bei der Darstellung wird das „A“ von „Aquarien“ unterstrichen dargestellt und der Benutzer kann mit Hilfe von „Alt-A“ die Checkbox fokussieren. |

Das folgende Beispiel enthält einen formatierten Text innerhalb eines `<bw:label>`-Elements.

```
<bw:label class="paragraph">
  Bitte geben Sie die gewünschten Suchbedingungen ein und clicken
  anschließend auf die Schaltfläche <i>Suchen</i>. Weitere
  Informationen erhalten Sie in der <link cmd="S_HELP">Suchhilfe
  </bw:link>.
</bw:label>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

BISS

## 9.6 Datenbindung

Um sinnvoll zu sein, müssen Dialoge in die Lage versetzt werden, Daten der Business-Tier (bzw. die Datenelemente des darin enthaltenen Objektmodells) anzeigen und Benutzereingaben entgegennehmen zu können.

Dies erfordert eine „Anbindung" der Dialogelemente (z.B. eines Eingabefeldes) an ein Attribut der Business-Tier bzw. des entsprechenden Objektmodells und umgekehrt. Bei den Attributen des Objektmodells kann es sich um einfache (skalare) Attribute wie z.B. den Nachnamen oder das Geburtsdatum des Kunden handeln oder um komplexe (strukturierte bzw. zusammengesetzte) Attribute wie z.B. die Liste aller Kunden, deren Nachname mit „M" beginnt.

Diese Anbindung erfolgt mit Hilfe des datasrc-Attrbuts (siehe Abschnitt 9.3.1.7) sowie mit Hilfe des `<bw:var>`-Tags (siehe auch Abschnitt 3.14) bzw. mit Hilfe des `element`-Attributs (siehe auch Abschnitt 3.3). Das `datasrc`-Attribut definiert dasjenige Datenelement (Attribut) innerhalb des Business-Objektmodells, dessen Inhalte innerhalb des entsprechenden Widgets angezeigt werden. Das `element`-Attribut bezieht sich immer auf Listenelemente und definiert diejenige Liste, auf deren Attribute anschließend zugegriffen werden soll.

Beispiele finden sich bei der Beschreibung der einzelnen Widgets.

Innerhalb des `expr`-Attributs des `<bw:var>`-Tags können Formatanweisungen verwendet werden, welche bei der Ausgabe der entsprechenden Daten verwendet werden. Die Formatanweisungen werden bei der Eingabe ignoriert. Sollen bei der Eingabe ggf. Filterregeln verwendet werden, so müssen andere, Widget-spezifische Attribute verwendet werden (z.B. das `mask`-Attribut des `<bw:entry>`-Elements).

Eine alternative Form der Datenanbindung ist eine Assoziation der Elemente mit den zugehörigen DOM-Werten (auf Serverseite) mit Hilfe entsprechender Methoden auf dem Server. Hierbei wird die (eindeutige!) ID des jeweiligen BML-Elements verwendet, um eine Verbindung mit dem DOM-Element auf Serverseite herzustellen. Eine typische Methode dieser Art sieht wie z.B. folgt aus:

```
protected void assocSearchCtrls () {
  LL_Navi     = assocCtrl( DLvl, "LL01");
  TP_Info     = assocCtrl( DLvl, "TP01");
  CH_QSrch    = assocCtrl( DLvl, "TB01.CH01");
  BU_Previous = assocCtrl( DLvl, "BU_Previous");
}
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Die DOM-seitigen Variablen (z.B. „LL_Navi“) sind vom Typ „ControlRef“, was u.a. eine automatische Propagation der Zustände wie „required“ oder „erroneous“ vom Server zum Client erlaubt.

## 9.7 Daten-Formatierung

Alle Elemente, die mit Hilfe des `datasrc`-Attributs mit einer Variablen des *B-Wise* Objektmodells assoziiert werden können, haben die Möglichkeit, optional mit Hilfe des Attributs `format` in ihrer Darstellung beeinflusst (formatiert) zu werden. Das `format`-Attribut gilt sowohl für den SmartClient als auch für den BrowserClient.

Zusätzlich kann für den SmartClient noch das `mask`-Attribut verwendet werden, welches eine Beeinflussung der Formatierung bzw. Filterung der Eingabe erlaubt (dabei werden bereits während der Eingabe ungültige Zeichen weggefiltert bzw. nicht zugelassen).

Werden beide Attribute angegeben, so gilt folgende Logik:

- Handelt es sich bei dem an der Session angemeldeten Client um den SmartClient und wurden beide Attribute (format und mask) verwendet, wird ausschließlich das mask-Attribut herangezogen und das format-Attribut ignoriert
- Handelt es sich bei dem an der Session angemeldeten Client um den BrowserClient und wurden beide Attribute (format und mask) verwendet, wird ausschließlich das format-Attribut herangezogen und das mask-Attribut ignoriert
- Handelt es sich bei dem an der Session angemeldeten Client um den BrowserClient und wurde nur das mask-Attribut verwendet, wird es ignoriert

Das `format`-Attribut enthält einen String bestehend aus einem Typ-Präfix und einem Format-String, der dem in java.text.SimpleDateFormat bzw. java.text.DecimalFormat entspricht. Das folgende Code-Fragment zeigt die Verwendung des `format`-Attributs bei einem Entryfield (Text-Eingabefeld):

```
<bw:entry id="TF_VersSumme" datasrc="Hausrat.VersSumme"
     format="C#,##0 EUR" > ... </bw:entry>
```

Die möglichen Präfixe enthält die folgende Tabelle:

| Präfix | Bedeutung |
|---|---|
| `C` | Currency |
| `B` | Timestamp |
| `D` | Date |
| `N` | Number |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `T` | Time, derzeit nicht unterstützt |
| `S` | String, derzeit nicht unterstützt |

Ist im *B-Wise* Repository bereits eine Typ-Information zum angebundenen Element vorhanden, wird diese ebenfalls herangezogen (gilt derzeit nur für gemirrorte Objekte bzw. Produkte).

Die möglichen Format-Zeichen für die Präfixe C und N zeigt die folgende Tabelle (soll eines der Zeichen als Literal im Prefix oder Suffix verwendet werden, muss es in einfache Hochkommata gesetzt werden, also z.B. `'%'` für die Ausgabe eines Prozentzeichens als Konstante):

| Symbol | Location | Loc. | Meaning |
|---|---|---|---|
| `0` | Zahl | Ja | Ziffer |
| `#` | Zahl | Ja | Ziffer, Null wird unterdrückt |
| `.` | Zahl | Ja | Dezimaltrennzeichen oder Dezimaltrennzeichen für Währung |
| `-` | Zahl | Ja | Minus-Zeichen |
| `,` | Zahl | Ja | Gruppen-Trennzeichen |
| `E` | Zahl | Ja | Trennt die Mantisse und den Exponnetne in wissenschaftlicher Notation |
| `;` | Teil-ausdruck-Grenze | Ja | Trennt positive und negative Teilausdrücke |
| `%` | Prefix oder Suffix | Ja | Mit 100 multiplizieren und als Prozentwert anzeigen |
| `\u2030` | Prefix oder Suffix | Ja | Mit 1000 multiplizieren und als Promillewert anzeigen |
| `¤` (`\u00A4`) | Prefix oder Suffix | Nein | Währungssymbol; wird in der Ausgabe durch das jeweilige Währungssymbol ersetzt. Wird es doppelt angegeben, wird das internationale Währungssymbol verwendet. Kommt es in einem Ausdruck vor, wird das dezimale Währungstrennzeichen anstatt des normalen Trennzeichens verwendet. |
| `'` | Prefix oder Suffix | Nein | Wird für die Kennzeichnung der o.g. Zeichen genutzt, um diese als Literale auszugeben; also z.B. `"'#'#"` formatiert 123 in `"#123"`. um ein einfaches Hochkomma auszugeben, sollte dieses 2x im Ausdruck vorkommen, z.B. `"# o''clock"`. |

Die möglichen Formate für die Präfixe B und D zeigt die folgende Tabelle:


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| Buchst. | Datum oder Zeitkomp. | Präsentation | Beispiele |
|---|---|---|---|
| G | Ära Kennzeichen | Text | AD |
| y | Jahr | Jahr | 1996; 96 |
| M | Monat im Jahr | Monat | July; Jul; 07 |
| w | Woche im Jahr | Zahl | 27 |
| W | Woche im Monat | Zahl | 2 |
| D | Tag im Jahr | Zahl | 189 |
| d | Tag im Monat | Zahl | 10 |
| F | Wochentag im Monat | Zahl | 2 |
| E | Wochentag | Text | Tuesday; Tue |
| a | am/pm Kennzeichen | Text | PM |
| H | Stunde am Tag | Zahl | 0 |
| k | Stunde am Tag (1-24) | Zahl | 24 |
| K | Stunde am Tag als am/pm (0-11) | Zahl | 0 |
| h | Stunde als am/pm (1-12) | Zahl | 12 |
| m | Minute einer Stunde | Zahl | 30 |
| s | Sekunde in einer minute | Zahl | 55 |
| S | Millisekunde | Zahl | 978 |
| z | Zeitzone | Generische Zeitzone | Pacific Standard Time; PST; GMT-08:00 |
| Z | Zeitzone | RFC 822 Zeitzone | -0800 |

Das `mask`-Attribut enthält einen String mit Zeichen, die folgende Bedeutung haben:

| Zeichen | Beschreibung bzw. Bedeutung |
|---|---|
| ToDo | ToDo. |

# 9.8 Elemente

Der folgende Abschnitt beschreibt alle in BML (Version 4.03) verfügbaren Widgets (Control-Elemente).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.1 <bw:label>

### Kurzbeschreibung

Ein `<bw:label>` ist ein Textelement zur Darstellung von Texten mit einer Reihe spezifischer Eigenschaften. So kann ein rechteckiger Bereich definiert werden, innerhalb dessen der Labeltext je nach Attribut-Definition umbricht oder abgeschnitten wird (in letzterem Fall werden 3 Punkte als Hinweis auf den gekürzten Text dargestellt). Zusätzlich kann einem Label ein PopUp-Hilfetext sowie ein Command zugewiesen werden, das bei Anclicken des Labeltextes ausgeführt wird. Ein Labeltext kann einfache Formatanweisungen enthalten, die eine entsprechende Darstellung des Textes ermöglichen (siehe auch Abschnitt 9.5). Zusätzlich kann ein Bezugs-Attribut (`for`) angegeben werden, das ein Label mit einem Widget (z.B. einem Eingabefeld) assoziiert. In diesem Fall wird beim Fokussieren des Widgets ein gepunktetes Rechteck um den Labeltext angezeigt.

ID-Konvention: „LA_“.

### Syntax

```
<bw:label
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      align="enum_alignment"
      for="str_widget_id"
      datasrc="bom_element"
      >
      <bw:content>
            Labelinhalt
      </bw:content>
</bw:label>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind: |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `align (E)` | (Optional) Definiert die Ausrichtung des Label-Textes innerhalb des zugehörigen Rechtecks. Erlaubte Werte sind „`TopLeft`“, „`TopRight`“, „`BottomLeft`“, „`MottomRight`“, „`MiddleLeft`“ und „`MiddleRight`“. |
| `for (S)` | (Optional) Definiert die ID eines Widgets, welches mit dem Label assoziiert ist. Wird eine gültige Assoziation hergestellt, wird beim Click auf das Label (den Text) das entsprechende Widget fokussiert; wird das assoziierte Widget fokussiert, wird um den Text des Labels ein gepunktetes Recht angezeigt. |
| `ellipsis (B)` | (Optional) Definiert, falls „true“, dass Textinhalte des Labels, die länger sind als der zur Verfügung stehende Platz, abgeschnitten werden; außerdem werden als Hinweis auf dieses Abschneiden 3 Punkte („Ellipsis“) hinter dem Text angezeigt. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck stehen, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

### Beispieldarstellung

```
<bw:label id="mylabel">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="30"/>
  <bw:style>
    <bw:font>Times Roman, 10, bold</bw:font>
    <bw:backgroundcolor>white</bw:backgroundcolor>
  </bw:style>
  <bw:events>
    <bw:event name="clicked" method="{event( 'doThis')}" />
  </bw:events>
  <bw:tooltip>
    <bw:tth><bw:t>Hinweis</bw:t></bw:tth>
    <bw:ttb><bw:t>Dies ist ein einfaches Label</bw:t></bw:ttb>
  </bw:tooltip>
  <bw:content>
    <bw:t>Dies ist ein <bw:b>langer</bw:b> Text, der innerhalb der
    Rechteck-Grenzen des Labels an Wortgrenzen (definiert durch
    Whitespace) umgebrochen und als mehrzeiliger Text dagestellt
    wird. Der Text enthält einige wenige <bw:i>Formatierungen</bw:i>
    </bw:t>.
  </bw:content>
</bw:label>
```

Selbstverständlich kann als Inhalt des Label-Elements (d.h. genauer als Inhalt des <bw:content>-Subelements) auch ein `<bw:var>`-Tag für die Anzeige von Texten verwendet werden. Der Inhalt des durch das `<bw:var>`-Tag bezeichneten Business-

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Objektmodell-Attributs wird gemäß den im `<bw:var>`-Tag definierten Regeln in Text konvertiert und an die entsprechende durch das Label definierte Stelle im Layout ausgegeben.

Die folgende Abbildung enthält einen Dialog mit einigen typischen Label-Elementen (bitte beachten Sie, dass die Linie ein eigenständiges Deco-Element ist und nicht zum Label „Partner suchen“ dazu gehört).

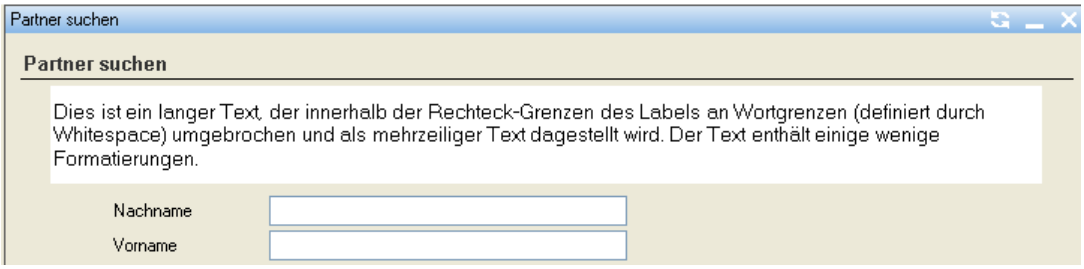

Partner suchen
Partner suchen
Dies ist ein langer Text, der innerhalb der Rechteck-Grenzen des Labels an Wortgrenzen (definiert durch Whitespace) umgebrochen und als mehrzeiliger Text dagestellt wird. Der Text enthält einige wenige Formatierungen.
Nachname
Vorname

**Abb. 17: Beispiele für Label-Elemente**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.2 <bw:htmllabel>

### Kurzbeschreibung

**Hinweis:** Dieses Element benötigt innerhalb des Clients das entsprechende PlugIn!

Ein `<bw:htmllabel>` ist ein Textelement zur Darstellung formatierter Texte mit einer Reihe (gegenüber dem `<bw:label>`-Element) erweiterter Eigenschaften. So können mit Hilfe von Tags (analog HTML) Textpassagen formatiert, Bilder eingefügt sowie bestimmte Darstellungseffekte erzielt werden. Zusätzlich können einem HTMLLabel ein PopUp-Hilfetext sowie einzelnen Textabschnitten des Gesamttextes Commands zugewiesen werden, die bei Anclicken der betreffenden Textpassagen ausgeführt werden.

ID-Konvention: „HL_“.

### Syntax

```
<bw:htmllabel
      id="str_widget_id"
      tabindex="int_tabindex"
      state="enum_state"
      visible="bool"
      >
      <bw:content>
            HTML-formatierter Labelinhalt
      </bw:content>
</bw:label>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwen- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | det und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.6 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.6.4). |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

Die folgende Tabelle enthält die verfügbaren Formatanweisungen, die innerhalb des Contents verwendet werden können.

**Schriftarten und Stile**

| Funktion | Tag | Attribute | Erlaubte Attributwerte | Beispiel |
|---|---|---|---|---|

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | | | | | |
|---|---|---|---|---|---|
| Schriftart | <font> … </font> | face | Name der Schriftart | Der Font muss mit dem Zielsystem kompatibel sein. | `face="Arial"` |
| | | size | Größe | Suffix „pt“ entspricht Pica, Suffix „px“ entspricht Pixel.Werte ohne Suffix und größer als 7 werden als Picas behandelt, Werte von 7 oder weniger werden in Standard HTML-Größen umgewandelt. | `size="12pt"` |
| | | color | Farbe | Eine der 17 in HTML erlaubten Namen oder ein RGB-Wert in hex in der Form "#RRGGBB" oder "rgb(RR,GG,BB)" oder "rgb(RR%,GG%,BB%)" | `color="green"`<br>`color="#008000"`<br>`color="rgb(00,80,00)"`<br>`color="rgb(0%,50%,0%)"` |
| Superscript Text | <sup> … </sup> | | | | `<sup>hoch</sup>` |
| Subscript Text | <sub> … </sub> | | | | `<sub>tief</sub>` |
| Font um ein Drittel vergrößern | <big> … </big> | | | | `<big>groß</big>` |
| Font um ein Drittel verkleinern | <small> … </small> | | | | `<small>klein</small>` |
| Fett |  …  | | | | `<b>fett</b>` |
| Kursiv |  …  | | | | `<i>kursiv</i>` |
| Unterstrichen | <u> … </u> | | | | `<u>unterstrichen</u>` |
| Durchgestrichen | <s> … </s> | | | | `<s>durchgestrichen</s>` |
| Hyperlink | <a> … </a> | href | Funktion | Ruft die entsprechende Funktion im Client oder Server auf. | `<a href="func()">Call </a>` |

## Layout

| Funktion | Tag | Attribute | Attributwerte | Beispiel |
|---|---|---|---|---|


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | | | | | |
|---|---|---|---|---|---|
| Umbruch | <br /> | | | | |
| Absatz (mit je einer Leerzeile ober- und unterhalb) |  …  | align | "left" (default) | Linksbündige Darstellung | `<p align="left">blah</p>` |
| | | | "right" | Rechtsbündige Darstellung | `<p align="right">blah</p>` |
| | | | "center" | Zentrierte Darstellung | `<p align="center">blah</p>` |
| | | | "justify" | Blocksatz | `<p align="justify">blah</p>` |
| Absatz (ohne Leerzeilen ober- und unterhalb) |  …  | align | "left" (default) | Linksbündige Darstellung | `<div align="left">blah</div>` |
| | | | "right" | Rechtsbündige Darstellung | `<div align="right">blah</div>` |
| | | | "center" | Zentrierte Darstellung | `<div align="center">blah</div>` |
| | | | "justify" | Blocksatz | `<div align="justify">blah</div>` |

## Images

| Funktion | Tag | Attribute | Attributwerte | | Beispiel |
|---|---|---|---|---|---|
| Bild anzeigen | <img> | src | Dateiname | Das Image wird immer vom Server geladen ! | `src="Image.jpg"` |
| | | align | "middle" | Die vertikale Mitte des Bildes wird an der Basiszeile des Textes ausgerichtet | |
| | | | "top" | Das obere Ende des Bildes wird am oberen Ende des Textes ausgerichtet | |
| | | | "bottom" (default) | Das untere Ende des Bildes wird an der Basiszeile des Textes ausgerichtet (Standardeinstellung) | |
| | | | "left" | Das Bild wird am linken Rand am Ende der Zeile ausgerichtet. | |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | | |
|---|---|---|
| | "right" | Das Bild ist am rechten Rand am Ende der Zeile ausgerichtet. |
| width | Breite des Bildes in Pixel | `width="50"` |
| height | Höhe des Bildes in Pixel | `height="50"` |
| border | Randbreite in Pixel | `border="2"` |
| vspace | Leerraum ober- und unterhalb des Bildes in Pixel | `vspace="10"` |
| hspace | Leerraum links und rechts neben dem Bild in Pixel | `hspace="10"` |

## Diverses

| Funktion | Tag | Anmerkungen |
|---|---|---|
| Ignoriere tags innerhalb des Blocks | <xmp> … </xmp> | <xmp> definiert einen Block, in dem alle Tags außer von </xmp> selbst ignoriert werden. Es erzwingt ebenso einen Zeilenumbruch. |

## Attribute, die an jedem Tag verwendet werden können

| Funktion | Attribut | Anmkerungen | Beispiel |
|---|---|---|---|
| Hintergrund-Farbe | bgcolor | Definiert die Hintergrundfarbe (erlaubte Angaben s.o.) | `<p bgcolor="yellow">...</p>` |
| Tooltip | title (oder tooltip) | Definiert einen Tooltip (auch BubbleHelp genannt), der bei mouseover erscheint. | `<b title="Text to appear in Tooltip">...</b>` |

Spezifische (Nicht-HTML-) Tags (**ACHTUNG:** diese stehen im BrowserClient nicht zur Verfügung !)

| Funktion | Tag | Attribute | Attributwerte | Beispiel |
|---|---|---|---|---|
| Gaussian Blur. Verfremdet den Text in einer definierten Form. | <blur> … </blur> | color | Farbe des Textes | `<blur color="silver" radius="4" xoffset="3" yoffset="3">text with shadow</blur>` |
| | | radius (oder 'r') | Radius des blur Effect in Pixel, je größer, umso unschärfer wird der Text. | |
| | | xoffset (oder 'x') | x-offset des Schattentextes | |
| | | yoffset (oder 'y') | y-offset des Schattentextes | |

Verwendet, wenn ein Schatteneffekt dargestellt werden soll.



© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Beispieldarstellung

```
<bw:htmllabel id="HL_1">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
       eox="165" eoy="65" ecx="500" ecy="300" />
  <bw:prop name="TextRenderingHint" value="ClearTypeGridFit" />
  <bw:prop name="WrapWordsOnly" value="true" />
  <bw:prop name="WordWrapDelimiters" value=" -" />
  <bw:prop name="UnderlineLinks" value="false" />
  <bw:prop name="TabStopDistance" value="50" />
  <bw:prop name="BorderStyle" value="FixedSingle" />
  <bw:content>
    <p><font face="Verdana" size="12" color="#ff0000">
       <b>Erklärende Hinweise</b></font></p>
       <p><font face="Tahoma">Dies ist das neue <b>HTML-Label</b>
       welches die Formatierung der Texte mit Hilfe von HTML-Tags
       erlaubt. Es unterstützt neben <b>bold</b> auch <i>italics</i>
       und unterschiedliche Fonts. Zudem werden Hyperlinks und noch
       was unterstützt, die u.a. auf Scripts gemappt werden können.
       So führt die <a href="cjscript://showBox( @id)"
       title="Triggert ein Script">Action</a> ein einfaches
       Script aus, welches eine Message-Box anzeigt.</font></p>
       <p>Dies ist ein neuer Absatz, im folgenden eine Liste von
       Tabs:</p>
       <div>Zelle 1,1<tab />Zelle 1,2<tab />Zelle 1,3<tab /></div>
       <div>Zelle 2,1<tab />Zelle 2,2<tab />Zelle 2,3<tab /></div>
       <div>Zelle 3,1<tab />Zelle 3,2<tab />Zelle 3,3<tab /></div>
       <div>Zelle 4,1<tab />Zelle 4,2<tab />Zelle 4,3<tab /></div>
  </bw:content>
</bw:htmllabel>
```

Selbstverständlich kann als Inhalt des HTMLLabel-Elements (d.h. genauer als Inhalt des `<bw:content>`-Subelements) auch ein `<bw:var>`-Tag für die Anzeige von Texten verwendet werden. Der Inhalt des durch das `<bw:var>`-Tag bezeichneten Business-Objektmodell-Attributs wird gemäß den im `<bw:var>`-Tag definierten Regeln in Text konvertiert und an die entsprechende durch das Label definierte Stelle im Layout ausgegeben.

Die folgende Abbildung enthält einen Dialog mit einem typischen HTMLLabel-Element. Die Abbildung entstand mit Hilfe der o.g. Beispiel-Definition (Quellcode).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

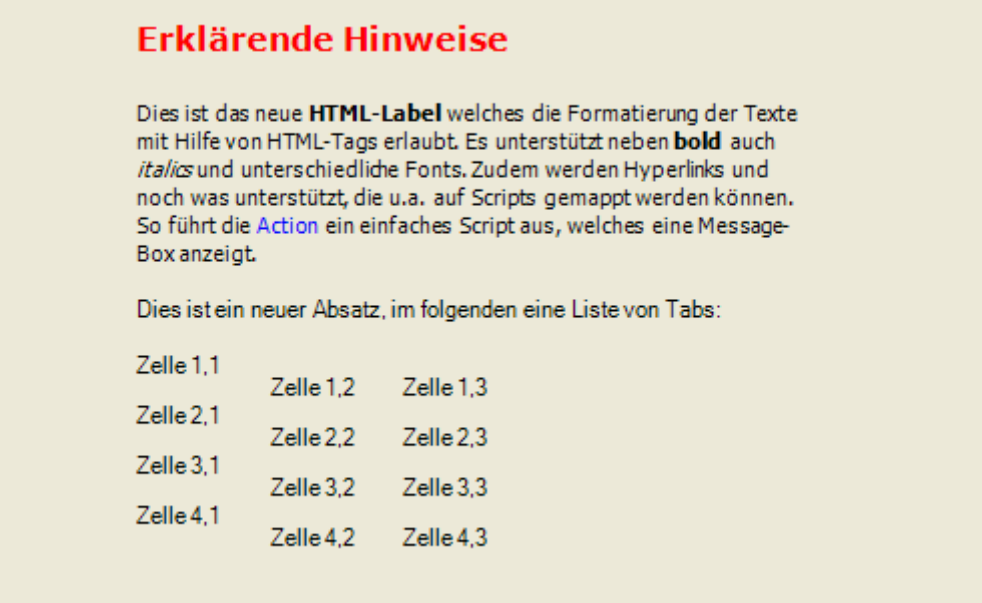

Erklärende Hinweise
Dies ist das neue HTML-Label welches die Formatierung der Texte mit Hilfe von HTML-Tags erlaubt. Es unterstützt neben bold auch italics und unterschiedliche Fonts. Zudem werden Hyperlinks und noch was unterstützt, die u.a. auf Scripts gemappt werden können. So führt die Action ein einfaches Script aus, welches eine Message-Box anzeigt.
Dies ist ein neuer Absatz, im folgenden eine Liste von Tabs:
Zelle 1,1 Zelle 1,2 Zelle 1,3
Zelle 2,1 Zelle 2,2 Zelle 2,3
Zelle 3,1 Zelle 3,2 Zelle 3,3
Zelle 4,1 Zelle 4,2 Zelle 4,3

**Abb. 18: Beispiele für ein HTMLLabel-Element**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.3 <bw:button>

### Kurzbeschreibung

Ein `<bw:button>` ist ein Schaltknopf (Button), der beim Anclicken ein Command auslöst. Die Darstellung des Buttons kann sowohl textuell als auch grafisch oder in einer Kombination hiervon erfolgen.

ID-Konvention: „BU_".

### Syntax

```
<bw:button
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      def="bool"
      >
      <bw:content>
            Buttonlabel
      </bw:content>
      <bw:img src="image_src" />
</bw:button>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles". |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID" bezeichnet und verwen- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | det und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.6.4). |
| `def (B)` | (Optional) Definiert, falls „true“, dass der Button im sog. „Default“-Zustand dargestellt und bei Eingabe von „Enter“ getriggert wird. Die Art der Darstellung hängt von der Basisplattform ab. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | (Optional) Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |
| `<bw:img>` | (Optional) zeigt ein Image (Bild) als Inhalt des Buttons an. Werden sowohl <bw:content> als auch <bw:img> definiert, überlagern sich beide Inhalte, sofern nicht mit Hilfe spezifischer <bw:prop>-Attribute das Layout entsprechend festgelegt wird. Dies gilt jedoch ausschließlich für die .NET-Plattform! |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar" (falls „true" als Argument übergeben wird) bzw. auf „unsichtbar" (falls „false" als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**addEvents()**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**removeEvents()**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**setDefault()**

Definiert den Button als „Default"-Button (siehe oben in Tabelle mit Attributen)..

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

### Beispieldarstellung

```
<bw:button id="BU_Prev" tabindex="100">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="30"/>
  <bw:content>~Zurück</bw:content>
  <bw:events>
    <bw:event name="clicked" method="{event('B_PREV')}"/>
  </bw:events>
</bw:button>
```

Die folgende Abbildung zeigt den Teil eines Dialogs mit typischen Button-Elementen.

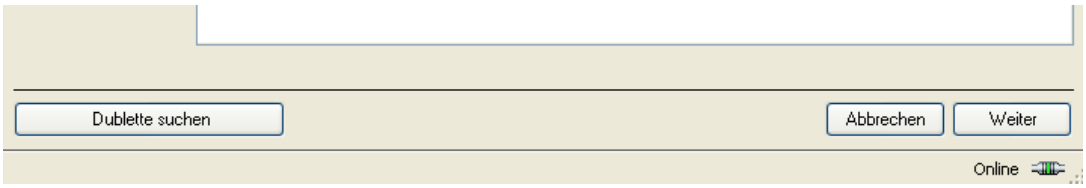



**Abb. 19: Beispiele für Button-Elemente**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.4 <bw:datetime>

### Kurzbeschreibung

Ein `<bw:datetime>` ist ein einzeiliges Auswahl-Element, welches die Auswahl bzw. Eingabe eines Datum sowie einer Uhrzeit ermöglicht.

ID-Konvention: „DP_“.

### Syntax

```
<bw:datetime
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      align="enum_alignment"
      datasrc="bom_element"
      >
      <bw:content>
            Widgetcontent
      </bw:content>
</bw:datetime>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informa- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | tionen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `spin (B)` | (Optional) Falls „true“, bestimmt, dass das Element als Spin-Button dargestellt werden soll; dabei wird nicht ein Monatkslaneder aufgeklappt, sondern der Wert einer Entität (Tag, Monat, Jahr, Stunde, Minute oder Sekunde) ausgewählt und mit Hilfe von Spin-Elementen inkrementiert oder dekrementiert; Default- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | Einstellung ist spin="false". |
| `format (S)` | Definiert das Format der Datum/Uhrzeit-Darstellung innerhalb des Elements; so zeigt z.B. format="dd.MM.yyyy hh:mm" das Datum entsprechend an. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>"-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags" gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip" innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true" oder „false")

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar" (falls „true" als Argument übergeben wird) bzw. auf „unsichtbar" (falls „false" als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**addEvents()**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**removeEvents()**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**setDateTime( datetime)**

Setzt den Wert des Datetime-Controls auf ‚datetime'.

**setMindateTime( mindatetime)**

Definiert das minimale Datum, das mit Hilfe des Controls gewählt werden kann.

**setMaxdateTime( maxdatetime)**

Definiert das maximale Datum, das mit Hilfe des Controls gewählt werden kann.

**setFormat( format)**

Setzt den Format-String (siehe auch Attribit „format" weiter oben).

**setSpinMode( boolvalue)**

Definiert den spin-Modus, falls „boolvalue" gleich „true" ist.

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**valuechanged [ val]**

Getriggert, wenn sich der Wert des Controls ändert (z.B. durch anklicken eines der Spinbuttons oder durch Auswahl eines Datums aus dem Monatskalender).

### Beispieldarstellung

```
<bw:datetime id="DP_Gebdat" datasrc="Partner.GebDat">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="192" ecy="32"/>
  <bw:tooltip>
    <bw:ttb>
      Bitte geben Sie hier das Geburtsdatum des Partners ein
    </bw:ttb>
  </bw:tooltip>
</bw:datetime>
```

Die folgende Abbildung enthält Datetime-Element mit offener (aufgeklappter) Auswahl.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

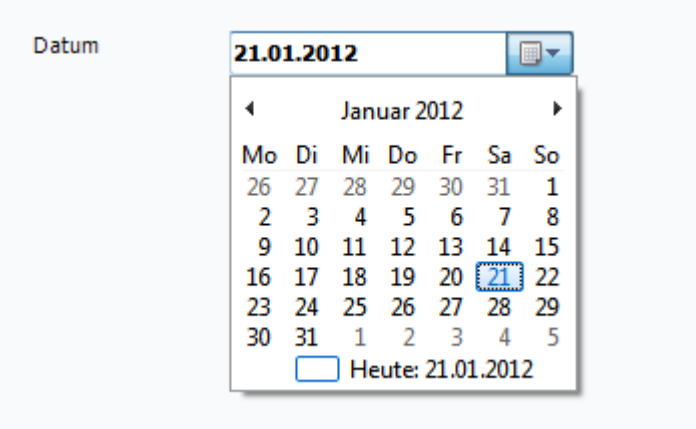



**Abb. 20: Beispiel für ein Datetime-Element (mit offener Auswahl)**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.5 <bw:mcalendar>

### Kurzbeschreibung

Ein `<bw:mcalendar>` ist ein Element, welches die Auswahl eines Datums ermöglicht.

**Hinweis:** Das `<bw:mcalendar>`-Element wird immer volständig dargestellt, unabhängig von der Angabe von Breite bzw. Höhe.

ID-Konvention: „MC_“.

### Syntax

```
<bw:mcalendar
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      scrollchange="int_val"
      date="str_date"
      datasrc="bom_element"
      >
      <bw:content>
            Widgetcontent
      </bw:content>
</bw:mcalendar>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets defi- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | niert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `scrollchange (I)` | (Optional) Definiert die Anzahl Monate, um die bei Klick auf einen der beiden |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | Scrollbuttons geblättert wird. |
| `date (S)` | (Optional) Definiert das Datum, das initial als markiert angezeigt wird; das Datum wird im Format „tt.mm.jjjj“ definiert, also z.B. „22.10.2009“. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)


© Copyright BISS GmbH, 2001-2012 All Rights reserved

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar" (falls „true" als Argument übergeben wird) bzw. auf „unsichtbar" (falls „false" als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`setDate( date)`**

Setzt das angezeigte Datum auf den Wert „date".

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**datechanged [ date ]**

Getriggert, nachdem das Datum (die Datumsauswahl eines einzelnen Datums) geändert worden ist.

**dateselected [ startdate, enddate ]**

Getriggert, nachdem ein Datumsbereich selektiert worden ist.

### Beispieldarstellung

```
<bw:mcalendar id="MC_GebDat">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="40" ecy="30"/>
  <bw:tooltip>
    <bw:ttb>
      Bitte wählen Sie das Geburtsdatum des Partners aus
    </bw:ttb>
  </bw:tooltip>
</bw: mcalendar >
```

Die folgende Abbildung enthält die Darstellung des <bw:mcalendar>-Elements.

**Abb. 21: Beispiele für ein Kalender-Element (<bw:mcalendar>)**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.6 <bw:entry>

### Kurzbeschreibung

Ein `<bw:entry>` ist ein einzeiliges Eingabefeld.

ID-Konvention: „TF_“.

### Syntax

```
<bw:entry
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      align="enum_alignment"
      datasrc="bom_element"
      password="bool"
      maxlen="lengh-in-characters"
      istart="index-no"
      iend="index-no"
      max="str_digits"
      format="str_format"
      >
      <bw:content>
            Widgetcontent
      </bw:content>
</bw:entry>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `format (S)` | (Optional) Definiert ein Format, das bei Ein- und Ausgabe der Werte verwendet wird. Beispiel: format=“ C#,##0.00 EUR“. Mögliche Werte siehe bitte Abschnitt 9.7. |
| `max (S)` | (Optional) Ein String bestehend aus Digits, z.B. „19999“; definiert die maximale Anzahl eingebbarer Zeichen und gleichzeitig deren maximalen Wert (Digits) |
| `password (B)` | (Optional) Falls „true“, werden all eingegebenen Zeichen als Sternchen (*) dargestellt; Default ist „false“. |
| `maxlen (I)` | (Optional) Definiert die maximale Länge der Eingabe in Zeichen; wurden maxlen Zeichen eingegeben, können keine weiteren Zeichen mehr eingegeben werden |
| `istart (I)` | (Optional) Definiert das erste Zeichen eines Selektionsbereichs |
| `iend (I)` | (Optional) Definiert das letzte Zeichen eines Selektionsbereichs |
| `align (E)` | (Optional) Definiert die Ausrichtung des Textes innerhalb des Eingabefeldes; mögliche Werte sind „left“, „right“ und „center“ |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`selectAll()`**

Selektiert den gesamten Inhalt des Eingabefeldes.

**`selectFormTo( begin, end)`**

Selektiert den Inhalt des Eingabefeldes von Zeichen „begin“ bis zum Zeichen „end“ (inklusive).

**`cut()`**

Schneidet den selektierten Inhalt aus und überträgt ihn ins Clipboard.

**`copy()`**

Kopiert den selektierten Inhalt ins Clipboard.

**`paste()`**

Fügt den Inhalt des Clipboards anstelle des selektierten Inhalts oder an die Cursor-Position.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**modified [ val ]**

Getriggert, nachdem der Wert des Eingabefeldes geändert worden ist.

### Beispieldarstellung

```
<bw:entry id="myentry" datasrc="Partner.Nachname" class="entrycls">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="30"/>
  <bw:tooltip>
    <bw:ttb>
      Bitte geben Sie hier den Nachnamen des Partners ein
    </bw:ttb>
  </bw:tooltip>
</bw:entry>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Die folgende Abbildung enthält einige typische Entryfields (hier u.a. für die Attribute „Straße und Haus-Nummer, PLZ oder Ort). Eingabefelder haben standardmäßig einen 1 Pixel breiten Rand.

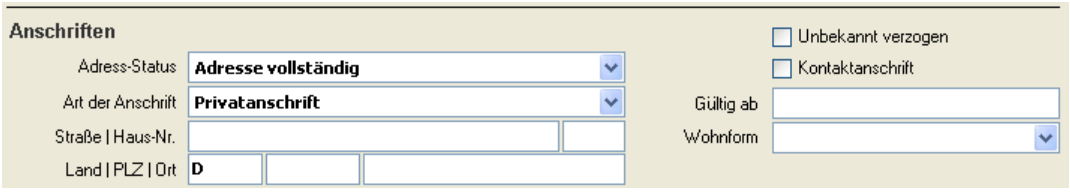



**Abb. 22: Beispiele für Eingabefelder (Entryfields)**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.7 <bw:editor>

### Kurzbeschreibung

Ein Editor ist ein Element zur Erfassung längerer, auch mehrzeiliger Texte. Seine Funktionalität ist derjenigen des Eingabefeldes sehr ähnlich.

ID-Konvention: „TE_“.

### Syntax

```
<bw:editor
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      datasrc="bom_element"
      istart="index-no"
      iend="index-no"
      readonly="bool"
      wrap="bool"
      >
      <bw:content>
            Widgetcontent
      </bw:content>
</bw:editor>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets defi- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | niert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `tab (B)` | (Optional) Falls „true“ legt fest, dass die TAB-Taste einen Tabulator in den Edi- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | tor einfügt anstatt von Feld zu Feld zu springen. Defalt ist „false“. |
| `wrap (B)` | (Optional) Falls „true“, wird der eingegebene Text automatisch am Ende einer Zeile umgebrochen. Default ist „false“. |
| `istart (I)` | (Optional) Definiert das erste Zeichen eines Selektionsbereichs |
| `iend (I)` | (Optional) Definiert das letzte Zeichen eines Selektionsbereichs |
| `scroll (E)` | (Optional) Definiert, ob der eingegebene Text innerhalb des Editors scrollen kann und damit sichtbar mehr Text eingegeben werden kann als sichtbar ist; mögliche Werte sind „none“, „vertical“, „horizontal“ und „both“; Default ist „vertical“. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`selectAll()`**

Selektiert den gesamten Inhalt des Eingabefeldes.

**`selectFormTo( begin, end)`**

Selektiert den Inhalt des Eingabefeldes von Zeichen „begin“ bis zum Zeichen „end“ (inklusive).

**`cut()`**

Schneidet den selektierten Inhalt aus und überträgt ihn ins Clipboard.

**`copy()`**

Kopiert den selektierten Inhalt ins Clipboard.

**`paste()`**

Fügt den Inhalt des Clipboards anstelle des selektierten Inhalts oder an die Cursor-Position.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

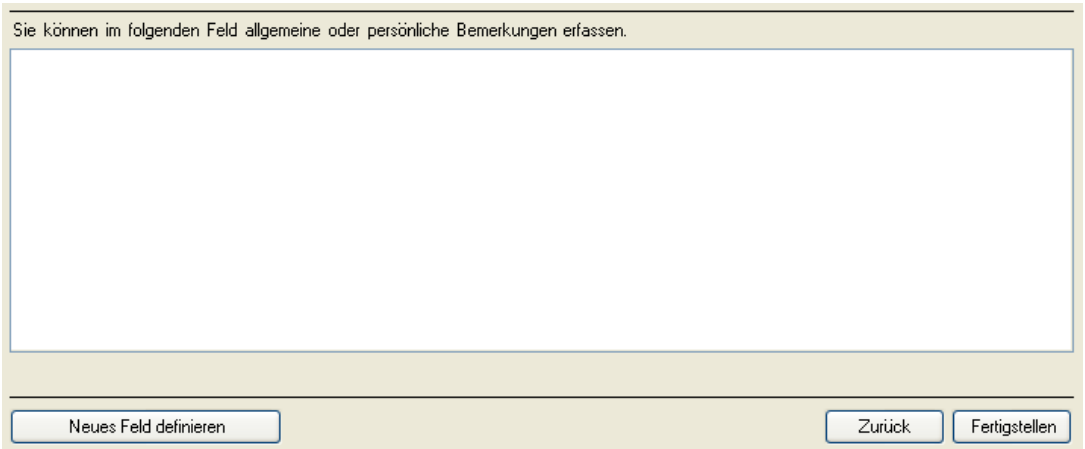
**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

### Beispieldarstellung

```
<bw:editor id="myedit" datasrc="Partner.Bemerkung" class="editcls">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="30"/>
</bw:editor>
```

Die folgende Abbildung zeigt einen Editor.





© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Abb. 23: Beispiel für einen Editor**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.8 <bw:texteditor>[3]

### Kurzbeschreibung

**Hinweis:** Dieses Element benötigt innerhalb des Clients das entsprechende PlugIn!

Ein Texteditor ist ein je nach Client dargestelltes Element zur Erfassung längerer, auch mehrzeiliger *formatierter* Texte. Als Format wird XHTML verwendet. Zum Styling der Texte kann CSS angewandt werden.

### Syntax

```
<bw:texteditor
      id="widget-id"
      datasrc="bom-element"
      class="css-class"
      tabindex="tabindex-no"
      istart="index-no"
      iend="index-no"
      >
      <bw:content>
            Widgetcontent
      </bw:content>
</bw:texteditor>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informa- |

[3] Dieses Widget ist aktuell in Entwicklung; stattdessen kann der <bw:editor> verwendet werden.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | tionen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `soezifisch (E)` | (Optional) Definiert die Ausrichtung des Label-Textes innerhalb des zugehörigen Rechtecks. Erlaubte Werte sind „`TopLeft`“, „`TopRight`“, „`BottomLeft`“, „`MottomRight`“, „`MiddleLeft`“ und „`MiddleRight`“. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsen- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | tiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)


© Copyright BISS GmbH, 2001-2012 All Rights reserved

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar" (falls „true" als Argument übergeben wird) bzw. auf „unsichtbar" (falls „false" als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Beispieldarstellung

```
<bw:texteditor id="myedit" datasrc="Partner.Bemerkung" class="editcls">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="30" />
  <bw:tooltip class="ttcls" width="200px">
    <bw:ttb>Sie können hier eine lange Bemerkung erfassen</bw:ttb>
  </bw:tooltip>
</bw:texteditor>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.9 <bw:syntaxeditor>

### Kurzbeschreibung

**Hinweis:** Dieses Element benötigt innerhalb des Clients das entsprechende PlugIn!

Ein `<bw:syntaxeditor>` ist ein je nach Client dargestelltes Element zur Erfassung von Programm-Quellcode. Innerhalb des SmartClients werden vielfältige Editierfunktionen, u.a. auch Syntax-Highlighting unterstützt. Der BrowserClient stellt den Syntaxeditor als gewöhnlichen Texteditor dar.

ID-Konvention: „SE_“.

### Syntax

```
<bw:syntaxeditor
      id="str_widget_id"
      tabindex="int_tabindex"
      enabled="bool"
      visible="bool"
      datasrc="bom_element"
      lines="bool"
      istart="index-no"
      iend="index-no"
      readonly="bool"
      syntax="enum_lang"
      >
      <bw:content>
            Widgetcontent
      </bw:content>
</bw:syntaxeditor>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `soezifisch (E)` | (Optional) Definiert die Ausrichtung des Label-Textes innerhalb des zugehörigen Rechtecks. Erlaubte Werte sind „`TopLeft`“, „`TopRight`“, „`BottomLeft`“, „`MottomRight`“, „`MiddleLeft`“ und „`MiddleRight`“. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)



© Copyright BISS GmbH, 2001-2012 All Rights reserved

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar" (falls „true" als Argument übergeben wird) bzw. auf „unsichtbar" (falls „false" als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Der Syntaxeditor verfügt über eine umfangreiche Funktionalität, die sich speziell an die Erfassung von Programmcode orientiert. Neben dem sog. Syntax-Highlighting, also der sprachspezifischen Einfärbung des Textes, unterstützt der Syntaxeditor beliebiges Undo (Rückgängig-Machen von Änderungen). Desweiteren werden u.a. folgende Eigenschaften bzw. Funktionen unterstützt:

- Unterstützung unterschiedlicher Programmiersprachen
- Reguläe Ausdrücke für die Suche und das Ersetzen von Texten
- Definition des Syntax-Highlightings in XML
- Lexikalisches Parsen des Textes in Tokenform
- Code-Outlining
- Unbegrenztes Undo/Redo
- Automatische Zeilennummern
- Nachvollziehen aller Zeilenänderungen
- Wrap-Modi
- Anzeige-Icons in der linken Spalte
- Optionale Anzeige von Leerzeichen
- Spaltenführungen
- Suche und Anzeige von Klammerpaaren
- Aufnahme und Wiedergabe von Tastatur-Makros
- Definierbare Funktionstasten
- Unterschiedliche Schriftarten und Schriftstile
- Suchen und Ersetzen
- 1-, 2- oder 4-Pane Anzeige
- WYSIWYG-Druck der Inhalte unmittelbar aus dem Editor
- HTML and RTF Export
- und vieles mehr...

Der Syntaxeditor eigent sich primär für spezielle *B-Wise* Prozesse aus dem Admin- bzw. Entwicklungs-Umfeld, wird jedoch extrem selten in Endbenutzer-Prozessen verwendet.

### Beispieldarstellung

```
<bw:syntaxeditor id="myedit" datasrc="Script" syntax="java"
lines="true">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="630"/>
</bw:syntaxeditor>
```

Die folgende Abbildung zeigt einen Editor. Der Rahmen muss explizit definiert werden, da der Editor selbst ohne Rahmen dargestellt wird.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

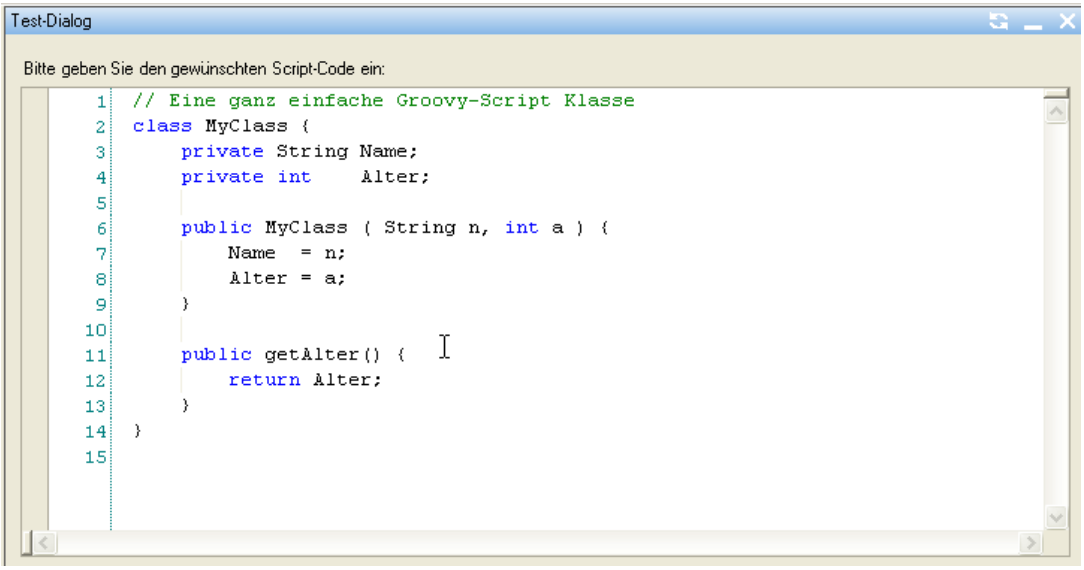



**Abb. 24: Beispiel für einen Syntax-Editor**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.10 <bw:browser>

### Kurzbeschreibung

Der `<bw:browser>` ist ein Element zur Anzeige von MIME-typisierten Inhalten. Innerhalb von *B-Wise* können damit aktuell XHTML-Seiten sowie PDF-Dokumente (über das Adobe Reader PlugIn) angezeigt werden. Das <bw:browser>-Element wird unter Windows auf den auf dem System installierten Standard-Internet Explorer (IE) abgebildet. Das gilt auch für die Version des IE, so dass entsprechende versionsabhängige Eigenschaften des IE berücksichtigt werden müssen.

Selbstverständlich können auch BML-Dialoge bzw. -Seiten innerhalb des `<bw:browser>`-Widgets angezeigt werden.

Der Inhalt des Browsers kann statisch definiert sein (normalerweise als BML-Datei, die mit Hilfe des `<bw:include>`-Tags eingefügt wird) oder dynamisch vom Prozess gesetzt werden (dann normalerweise mit Hilfe des `<bw:var>`-Tags eingefügt oder über das Server-API gesetzt).

ID-Konvention: „BR_“.

### Syntax

```
<bw:browser
      id="str_widget_id"
      tabindex="int_tabindex"
      enabled="bool"
      visible="bool"
      src="reference"
      usemenu="bool"
      >
      <bw:content>
            Browser-Content
      </bw:content>
</bw:browser>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind: |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `source (S)` | (Optional) Die URL der anzuzeigenden Seite. Im Ggs. zur Anmgabe eines <bw:content>-Elements wird die URL des Source-Attributs über eine http(s)-Verbindung in das Browser-Control geladen; der Content wird im Ggs. hierzu als DOM-Fragment eingefügt. |
| `usemenu (B)` | (Optional) Falls „true“, zeigt das PopUp-Menü des Browser-Controls an; dieses erlaubt u.a. Inhalte ins Clipboad zu kopieren. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl kon- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

stanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element).

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`setSrc( src)`**

Setzt das src-Attribut auf den als Argument übergebenen Wert und lädt gleichzeitig die darin enthaltene URL

**`back()`**

Entspricht dem Klick auf den Browser-Back-Button.

**`forward()`**

Entspricht dem Klick auf den Browsre-Forward-Button.

**`home()`**

Lädt die initial innerhalb des Browser-Controls angezeigt URL.

**`blank()`**

Löscht den Inhalt des browser-Controls und zeigt eine leere Seite an.

**`stop()`**

Unterbricht einen ggf. laufenden Ladevorgang einer URL.

**`print()`**

Zeigt den Druckdialog an und erlaubt damit den Ausdruck der aktuell angezeigten Seite.

**`printDefault()`**

Druckt sofort die gerade angezeigt Seite auf den Standard-Drucker (ohne Dialoganzeige).

**`printPreview()`**

Zeigt die Druckvorschau der aktuell angezeigten Seite an.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`pageSetup()`**

Zeigt den „Seite einrichten"-Dialog an.

**`properties()`**

Gibt die aktuellen Eigenschaften des Browser-Controls zurück

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**cangobackchanged []**

Getriggert, wenn sich der Wert des „cangoback"-Attributs ändert. Das cangoback-Attribut definiert, ob eine vorige Seite existiert bzw. angezeigt werden kann.

**documentcompleted []**

Getriggert, nachdem das Browser-Control das Laden einer Seite (URL) beendet hat.

**navigating []**

Getriggert, wenn das Browser-Controls begonnen hat, eine Seite (URL) zu laden.

### Beispieldarstellung

```
<bw:browser id="BR_Main" tabindex="1">
  <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
          eox="0" eoy="0" ecx="0" ecy="0"/>
  <bw:content>
    <bw:include src="{HtmlPage}" />
  </bw:content>
</bw:browser>
```

Die folgende Abbildung (Abb. 25) zeigt ein Browser-Widget mit einer vom Prozess definierten XHTML-Seite, die wiederum statische Informationen enthält.

Das `<bw:browser>`-Element wird insbesondere für die Anzeige von Informationen inkl. komplexer Listen und Tabellen verwendet. Die in XHTML gegebenen Möglichkeiten inkl. CSS-Formatierung und Dynamik mit Hilfe von Javascript bieten eine breite Unterstützung beim layout komplexer Anzeigeseiten sowie erforderlichen Dynamiken.

Es ist Aufgabe eines jeden einzelnen *B-Wise* Projekts, diesbezügliche Standards festzulegen, da die Variabilität von Projekt zu Projekt sehr hoch sein kann. Zu den festzulegenden Standards bzw. Konventionen gehören u.a.

- Browser-Version (ist es möglich, sich auf eine Minimal-Version zu stützen?); dies bestimmt insbesondere die grundsätzlichen Möglichkeiten wie z.B. Funktionsumfang, CSS-Styling, „Browser-Bugs" etc.
- Verwendung externer Bibliotheken (z.B. jQuery oder auch weirgehender, ggf. von der Basisbibliothken abgeleiteter Bibliotheken wie z.B. DataTables für die Tabellendarstellung)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

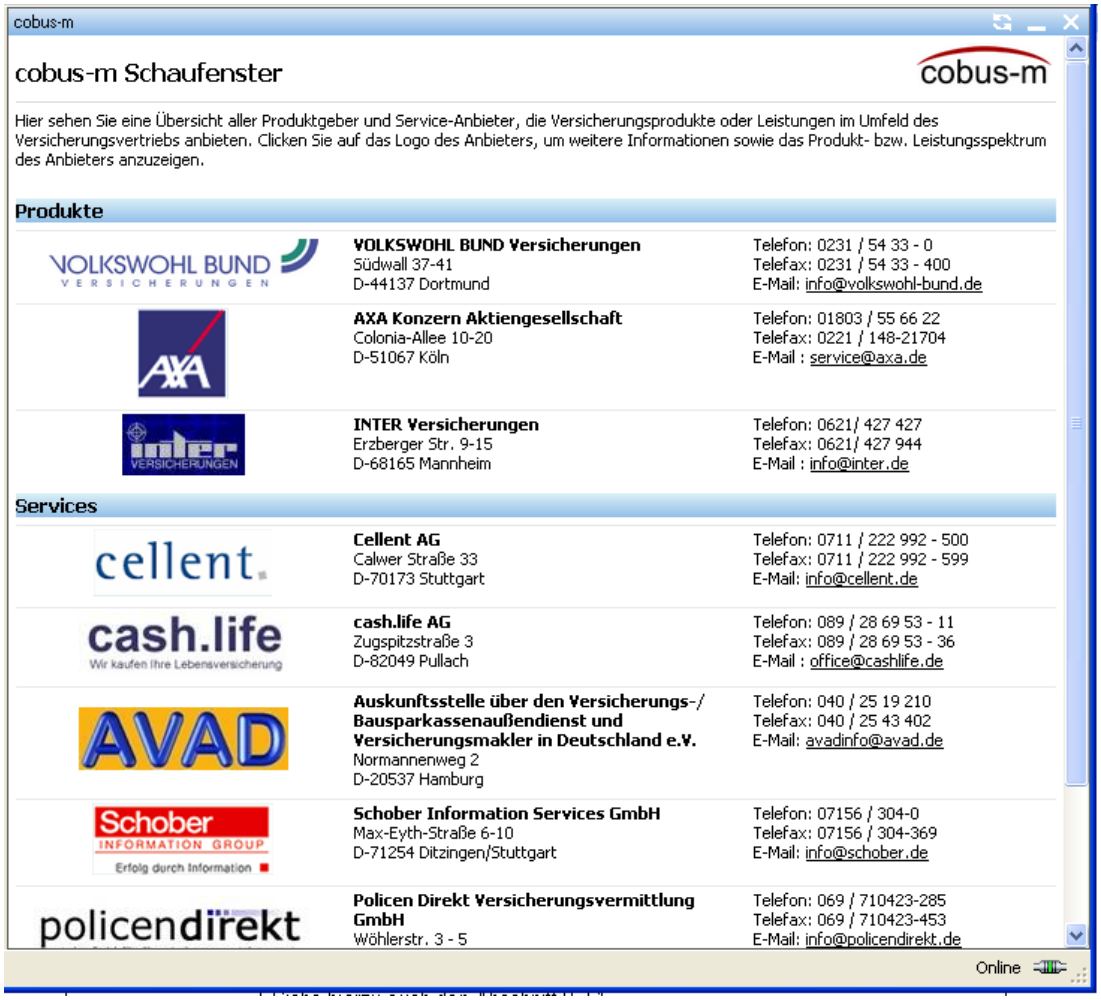

cobus-m
cobus-m Schaufenster
cobus-m
Hier sehen Sie eine Übersicht aller Produktgeber und Service-Anbieter, die Versicherungsprodukte oder Leistungen im Umfeld des Versicherungsvertriebs anbieten. Clicken Sie auf das Logo des Anbieters, um weitere Informationen sowie das Produkt- bzw. Leistungsspektrum des Anbieters anzuzeigen.
Produkte
VOLKSWOHL BUND Versicherungen
Südwall 37-41
D-44137 Dortmund
Telefon: 0231 / 54 33 - 0
Telefax: 0231 / 54 33 - 400
E-Mail: info@volkswohl-bund.de
AXA Konzern Aktiengesellschaft
Colonia-Allee 10-20
D-51067 Köln
Telefon: 01803 / 55 66 22
Telefax: 0221 / 148-21704
E-Mail : service@axa.de
INTER Versicherungen
Erzberger Str. 9-15
D-68165 Mannheim
Telefon: 0621/ 427 427
Telefax: 0621/ 427 944
E-Mail : info@inter.de
Services
Cellent AG
Calwer Straße 33
D-70173 Stuttgart
Telefon: 0711 / 222 992 - 500
Telefax: 0711 / 222 992 - 599
E-Mail: info@cellent.de
cash.life AG
Zugspitzstraße 3
D-82049 Pullach
Telefon: 089 / 28 69 53 - 11
Telefax: 089 / 28 69 53 - 36
E-Mail : office@cashlife.de
Auskunftsstelle über den Versicherungs-/ Bausparkassenaußendienst und Versicherungsmakler in Deutschland e.V.
Normannenweg 2
D-20537 Hamburg
Telefon: 040 / 25 19 210
Telefax: 040 / 25 43 402
E-Mail: avadinfo@avad.de
Schober Information Group
Max-Eyth-Straße 6-10
D-71254 Ditzingen/Stuttgart
Telefon: 07156 / 304-0
Telefax: 07156 / 304-369
E-Mail: info@schober.de
Policen Direkt Versicherungsvermittlung GmbH
Wöhlerstr. 3 - 5
Telefon: 069 / 710423-285
Telefax: 069 / 710423-453
E-Mail: info@policendirekt.de
Online

**Abb. 25: Beispiel für einen Browser**



© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.11 <bw:checkbox>

### Kurzbeschreibung

Eine `<bw:checkbox>` ist eine Schaltfläche, mit deren Hilfe boolsche (ja/nein-) oder auch dreiwertige (also ja/nein/unbekannt) Zustände definiert werden können. Wird die Checkbox an ein Element des Objektmodells gebunden, dann wird der „null"-Zustand dieses Elements als „unbekannt" angezeigt.

ID-Konvention: „CB_".

### Syntax

```
<bw:checkbox
   id="str_widget_id"
   styleref="str_style_reference"
   class="str_styleclass"
   tabindex="int_tabindex"
   state="enum_state"
   enabled="bool"
   visible="bool"
   textalign="enum_textalignment"
   checkalign="enum_checkalignment"
   checkstate="enum_state"
   datasrc="bom_element"
   >
   <bw:content>
     Labelinhalt
   </bw:content>
<bw:checkbox>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `textalign (E)` | (Optional) Definiert die Ausrichtung des Label-Textes im Verhältnis zum Kästchen der Checkbox; mögliche Werte sind eine Kombination aus „top“, „middle“, „bottom“ und „left“, „center“, „right“, z.B. „topleft“; Default ist „topleft“ |
| `checkalign (E)` | (Optional) Definiert die Position des Kästchens in Relation zum Text; mögliche Werte sind „left“, „right“, „topleft“ und „topright“ |
| `checkstate (E)` | (Optional) Definiert den Zustand der Checkbox; mögliche Werte sind „unchecked“, „checked“ und „indeterminate“ (für eine Three-state-checkbox) |
| `tri (B)` | (Optional) Falls „true“ definiert eine Three-state-checkbox, also eine Checkbox mit drei möglichen Zuständen |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**setStyleRef( styleref )**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**addEvents()**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**removeEvents()**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**setCheckState( state)**

Setzt den Zustand der Checkbox auf den Wert „state".

**setThreeState( bool)**

Falls bool="true" setzt die Checkbox in den Three-state-Zustand.

**setTextAlign( align)**

Setzt die Ausrichtung des Textes auf einen der erlaubten Werte (siehe oben).

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**`statechange [ checkstate ]`**

Getriggert, wenn die Checkbox beginnt, den Zustand durch den Anwender zu ändern.

**`statechanged [checkstate ]`**

Getriggert, nachdem die Checkbox den Zustand durch den Anwender geändert hat.

**`pstatechanged [checkstate ]`**

Getriggert, nachdem die Checkbox ihren zustand durch den Prozess oder ein Skript geändert hat.

### Hinweise

Wird eine Checkbox mit Hilfe des `datasrc`-Tags an ein Datenelement angebunden, welches nicht vom Typ boolean ist, werden bei der Anzeige alle Werte gleich 0 (null) oder Strings der Länge 0 als false (nicht gecheckt), alle anderen Werte als true (gecheckt) interpretiert. Bei der Bearbeitung liefert die Checkbox bei der Anbindung an ein numerisches Datenelement die Werte „0“ und „1“, bei der Anbindung an ein String-Datenelement die Werte „true“ und „false“.

### Beispieldarstellung

```
<bw:checkbox id="mycheckbox" datasrc="Product.Fahrraddiebstahl.Select"
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="30"/>
    <bw:t>Auch ~Aquarien versichern</bw:t>
  </bw:label>
  <bw:tooltip>
    <bw:ttb>Bitte geben Sie hier an, ob Sie auch Aquarien innerhalb
    der Hausratversicherung versichern möchten</bw:ttb>
  </bw:tooltip>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
</bw:checkbox>
```

Die folgende Abbildung zeigt einige Checkboxen in unterschiedliche Zuständen (markiert, nicht markiert und im Zustand „unbekannt“. Die gelbe Hintergrundfarbe markiert eine MUSS-Feld Checkbox.

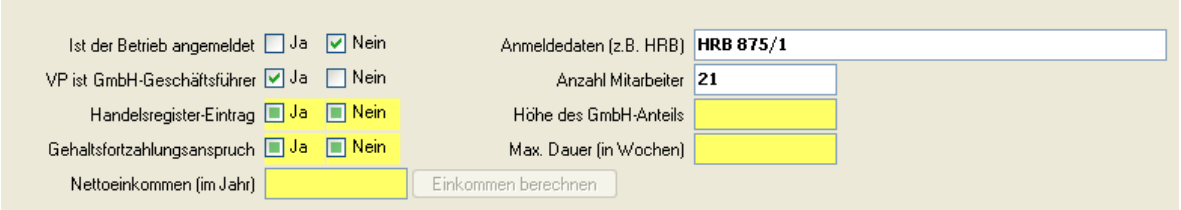



**Abb. 26: Beispiele für Checkboxen**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.12 <bw:choice>

### Kurzbeschreibung

Eine `<bw:choice>` ist ein (optional editierbares) Feld mit einer aufklappbaren Liste, aus der genau eine von mehreren Optionen gewählt werden kann. Innerhalb von Windows-Dialogen wird dieses Element auch ComboBox genannt.

ID-Konvention: „CH_“.

### Syntax

```
<bw:choice
  id="str_widget_id"
  styleref="str_style_reference"
  class="str_styleclass"
  tabindex="int_tabindex"
  state="enum_state"
  enabled="bool"
  visible="bool"
  datasrc="bom_element"
  edit="bool"
  select="int_index"
  sorted="bool"
  valselect="bool"
  >
  <bw:content>
    Widgetcontent
  </bw:content>
</bw:choice>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `edit (B)` | (Optional) Falls „true“, bestimmt, dass Werte nicht nur ausgewählt sondern auch eingegeben werden können; Default ist „false“. |
| `select (I)` | (Optional) Wählt das I-te Element der Auswahlliste als aktuellen Wert der Choice aus; der Index bezieht sich auf die Originalliste und nicht auf die ggf. sortierte Liste! |
| `sorted (B)` | (Optional) Falls „true“, werden die Inhalte der Auswahlliste alphabetisch aufsteigend sortiert |
| `valselect (B)` | (Optional) Falls „true“ wird beim „Durchsteppen“ der Liste durch den Anwender (also das nacheinander Selektieren eines Eintrags mit Hilfe der Tastatur oder Maus) jeweils ein validate-Event getriggert |
| `align (E)` | (Optional) Definiert die Ausrichtung des Inhalts innerhalb der Choice; mögliche Werte sind „left“, „center“ und „right“; Default ist „left“. |
| `padding (I)` | (Optional) Definiert ein vertikales Padding zwischen den Elementen der Auswahlliste; der Wert ist „non-collapsing“, d.h. zwischen den Werten wird der Wert addiert. |
| `value (S)` | (Optional) Initialer Anzeigewert bei nicht spezifiziertem „select“-Attribut |
| `maxvis (I)` | (Optional) Definiert die Anzahl maximal sichtbarer Zeilen wenn die Auswahlliste aufgeklappt ist |
| `acmode (E)` | (Optional) Definiert die Art, wie die Choice Tastatureingaben auf die Werte der Auswahlliste abbildet und dadurch einen Wert auswählt (AutoCompleteMode); mögliche Werte sind: „none“ (deaktiviert das Feature zum automatischen Vervollständigen), „suggest“ (zeigt die dem Edit-Steuerelement zugeordnete Hilfs-dropdownliste an. Diese Dropdownliste wird mit einer oder mehreren vorgeschlagenen Abschlusszeichenfolgen aufgefüllt), „append“ (fügt die verbleibenden Zeichen der am ehesten in Frage kommenden Zeichenfolge an die vorhandenen Zeichen an und markiert die angehängten Zeichen) und „suggestappend“ (kombinkiert die Werte von „append“ und „suggest“). |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`selectItem( item)`**

Selektiert das item-te Element der Auswahlliste als den aktuellen Wert der Choice.

**`addItems( items)`**

Fügt die items ans Ende der Auswahlliste an.

**`insertItems( items, position)`**

Fügt die items der Auswahlliste an die Position „position“ ein.

**`removeItems( from, to)`**

Löscht die Elemente der Auswahlliste beginnend mit Element from bis zum Element to.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**setText( text)**

Setzt den Text der Auswahlliste auf den Wert „text“.

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**selected [ pos ]**

Getriggert, nachdem vom Anwender ein Wert der Auswahlliste ausgewählt worden ist.

**textupdate []**

Getriggert, nachdem sich der Text der Auswahlliste geändert hat.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**pselectionchanged [ pos]**

Getriggert, nachdem sich die Auswahl durch den Prozess oder ein Skript geändert hat.

### Beispieldarstellung

```
<bw:choice id="mychoice" datasrc="Partner.Adresse.Status">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="20" eoy="10" ecx="220" ecy="30"/>
</bw:choice>
```

Die folgende Abbildung enthält einige typische Choices (hier u.a. für die Attribute „Adress-Status“, „Art der Anschrift“ und „Wohnform“). Choices haben standardmäßig einen 1 Pixel breiten Rand und eine automatische Anpassung der Höhe an die Schriftart. Letzteres ist eine Funktionalität von Windows, die bewusst in *B-Wise* nicht geändert worden ist. Diese Anpassung ist allerdings nur im SmartClient .net gegeben, andere Clients passen die Höhe nicht autoamtisch an.

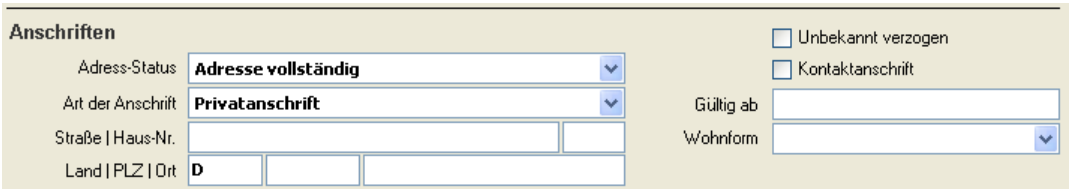



**Abb. 27: Beispiele für Choices (Comboboxen)**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.13 <bw:list>

### Kurzbeschreibung

Eine `<bw:list>` ist eine spaltenorientierte Liste mit einer (optionalen) Überschriften-Zeile. Mit Hilfe interner Attribute und Elemente sowie ggf. komplexer BML-Logik können umfangreiche Formatierungen durchgeführt werden.

### Syntax

```
<bw:list
  id="str_widget_id"
  styleref="str_style_reference"
  class="str_styleclass"
  tabindex="int_tabindex"
  state="enum_state"
  enabled="bool"
  visible="bool"
  checkstyle="bool"
  editable="bool"
  startidx="int_index"
  select="int_index"
  >
  <bw:content>
    <bw:lhead>
      <bw:lc width="str_width">
        Überschriften-Text
      </bw:lc>
    </bw:lhead>
    <bw:lgroups>
      <bw:lgroup id="grp_id">
        Gruppen-Label
      </bw:lgroup>
    </bw:lgroups>
    <bw:lbody>
      <bw:ldata element="om_list_ref">
        <bw:li id="int_item_id" group="grp_id">
          <bw:img />
          <bw:lc>
            Wert der Spalte
          </bw:lc>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
          </bw:li>
        </bw:ldata>
      </bw:lbody>
    </bw:content>
    <bw:events>
      <bw:event method="method" name="event_name"/>
    </bw:events>
</bw:list>
```

Das `<bw:lbody>`-Element bildet die Transformationseinheit, die dynamisch bei Updates der Liste übertragen wird.

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible="`true`") oder nicht (visible="`false`"); Standard-Einstellung ist visible="`true`". |
| `rolocked (B)` | (Optional) Wenn „true", verhindert Hittesting und die Erkennung von Events im „readonly"-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly"-Zustand auf „mouse-over"-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true", erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock"-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset") um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>"-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `checkstyle (B)` | (Optional) Falls „true", wird am Beginn (also links) einer jeden Zeile eine Checkbox dargestellt; Default ist „false" |
| `editable (B)` | (Optional) Falls „true", kann der Inhalt einer jeden Zeile innerhalb der Zeile bearbeitet (ediiert) werden; Default ist „false" |
| `sortable (B)` | (Optional) Falls „true", kann der Inhalt durch Klick auf die Überschrift auf- bzw. absteigend sortiert werden; Default ist „false". |
| `select (I)` | (Optional) Null-basierter Index, der das initial ausgewählte Element angibt; das Element wird initial selektiert dargestellt |
| `multisel (B)` | (Optional) Falls „true", ermöglicht die Auswahl von mehr als einer Zeile innerhalb der Liste; Default ist „false" |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerte- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | ten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**setRequired( state )**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**setState( state )**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**setFocus()**

Setzt den Focus auf das Label-Element.

**setVisible()**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**setCtxMenu()**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**setTooltip( item, tooltip )**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**setStyle( attrname, attrval )**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**setStyleRef( styleref )**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**addEvents()**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**removeEvents()**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**clear()**

Löscht die Selektion der Liste.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`removeItems( from, to)`**

Löscht die Elemente der Liste beginnend mit „from“ bis einschließlich „to“.

**`setItemsChecked( from, to, state)`**

Setzt bei einer mit Checkboxen ausgestatteten Liste die Elemente beginnend mit „from“ bis einschleßlich „to“ auf den „angehakt“-Status (state = „true“) bzw. „nicht-angehakt“(state = „false“).

**`setItemsSelected( from, to, state)`**

Selektiert (state = „true“) bzw. deselektiert (state = „false“) die Elemente beginnend mit „from“ bis einschleßlich „to“.

**`setItemsEnabled( from, to, state)`**

Aktiviert (state = „true“) bzw. deaktivier (state = „false“) die Elemente der Liste beginnend mit „from“ bis einschließlich „to“.

**`setIdTooltip( id, text)`**

Setzt den Tooltip-Text des Listenelements mit der ID „id“ auf den Wert „text“.

**`setItemsStyleRef( from, to, style)`**

Setzt die Darstellungsart (Style) der Elemente beginnen mit „from“ bis einschließlich „to“ auf den Style „style“.

**`addItems( items)`**

Fügt die Elemente „items“ ans Ende der Liste an.

**`insertItems( items, id)`**

Fügt die Elemente „items“ nach dem Element mit der ID „id“ ein.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**checkrequest []**

Getriggert, nachdem der Anwender eine entmarkierte (nicht angehakte) Checkbox markiert hat, aber bevor die Checkbox tatsächlich geändert wird.

**uncheckrequest []**

Getriggert, nachdem der Anwender eine markierte (angehakte) Checkbox entmarkiert hat, aber bevor die Checkbox tatsächlich geändert wird

**check []**

Getriggert, nachdem der Anwender eine entmarkierte (nicht angehakte) Checkbox markiert hat.

**uncheck []**

Getriggert, nachdem der Anwender eine markierte (angehakte) Checkbox entmarkiert hat.

**checkchanged [ checked ]**

Getriggert, nachdem der Zustand der Checkbox geändert worden ist.

**selected [ nid ]**

Getriggert, nachdem der Anwender ein Element der Liste selektiert hat.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**deselected []**

Getriggert, nachdem der Anwender ein Element der Liste deselektiert hat.

**selectionchanged [ nid ]**

Getriggert, nachdem sich der Zustand eines Elements geändert hat.

**itemclicked [ nid ]**

Getriggert, nachdem ein Element der Liste angeklickt worden ist.

**itemdoubleclicked []**

Getriggert, nachdem ein Element der Liste doppelt angeklickt worden ist.

**colsort [ pos, val ]**

Getriggert, nachdem die Sortierugn einer Spalte geändert worden ist.

**pselectionchanged [ nid, val ]**

Getriggert, nachdem die Auswahl (selektiertes Element) durch den prozess oder ein Skript geändert worden ist.

### Hinweise

Als Inhalte von `<bw:lc>`-Elementen sind Text bzw. alle Elemente zugelassen, die Text generieren (z.B. das `<bw:var>`-Element) sowie das `<bw:img>`-Element.

### Beispieldarstellung

```
<bw:list id="LI_Procs">
  <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
           eox="0" eoy="0" ecx="0" ecy="0"/>
  <bw:content>
    <bw:lhead>
      <bw:lc width="10%" />
      <bw:lc width="70%" />
      <bw:lc width="20%" />
    </bw:lhead>
    <bw:lgroups>
      <bw:foreach element="Processes.*ItemIfc">
        <bw:lgroup id="{.Category.Path.hashCode}">
          <bw:var expr=".Category.Path" />
        </bw:lgroup>
      </bw:foreach>
    </bw:lgroups>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    <bw:lbody>
      <bw:ldata element="Processes.*ItemIfc">
        <bw:if expr="isProcessAllowed( .ID)">
          <bw:li id="{concat( .Location.Name ,'|', .language)}"
                                  group="{.Category.Path.hashCode}">
            <bw:img src="images/process.gif"/>
            <bw:lc />
            <bw:lc><bw:var expr=".Name"/></bw:lc>
            <bw:lc><bw:var expr=".language"/></bw:lc>
          </bw:li>
        </bw:if>
      </bw:ldata>
    </bw:lbody>
  </bw:content>
  <bw:events>
    <bw:event method="callProcess" name="selected"/>
  </bw:events>
</bw:list>
```

Die folgende Abbildung zeigt eine typische Liste mit logischen Gruppen. Der entsprechende BML-Code ist in der obigen Beispiel-Definition wiedergegeben.

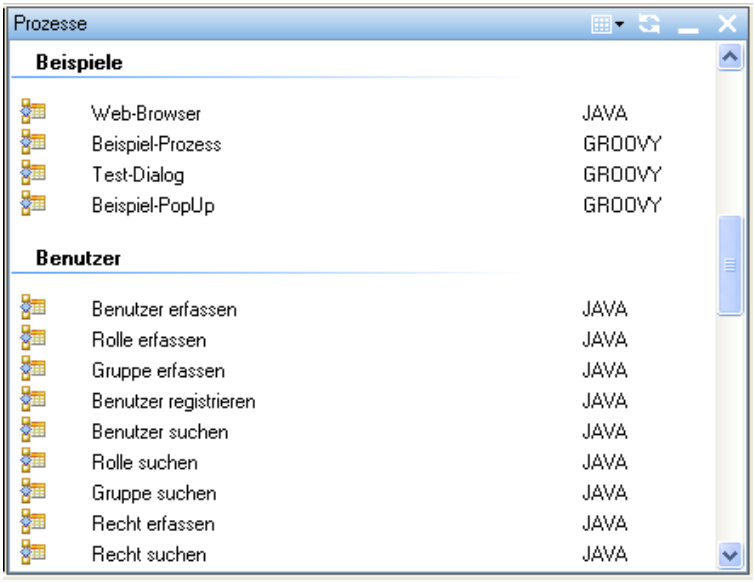



**Abb. 28: Beispiel einer Liste**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.14 <bw:datagrid>

### Kurzbeschreibung

Ein `<bw:datagrid>` ist eine Matrix von Zellen, die Texte, Zahlen sowie Controls (Entryfields, Checkboxen und Choices) enthalten können. In vielen Punkten entspricht es einem Spreadsheet. Das <bw:datagrid> ist ein komplexes Element und sollte daher sparsam verwendet werden, da die Ressourcen-Anforderungen rel. hoch sind.

ID-Konvention: „DG_“.

### Syntax

```
<bw:datagrid
  id="str_widget_id"
  styleref="str_style_reference"
  class="str_styleclass"
  tabindex="int_tabindex"
  state="enum_state"
  enabled="bool"
  visible="bool"
  readonly="bool"
  extendable="bool"
  autosize="enum_size"
  header="enum_head"
  >
  <bw:content>
    <bw:cols>
      <bw:col id="str_col_id" width="str_width" type="enum_type">
        Überschriften-Text
      </bw:col>
    </bw:cols>
    <bw:rows>
      <bw:row id="row_id">
        <bw:cell id="cell_id">
          Zellen-Inhalt
        </bw:cell>
      </bw:row>
    </bw:rows>
  </bw:content>
  <bw:events>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    <bw:event method="method" name="event_name"/>
  </bw:events>
</bw:list>
```

Das <bw:rows>-Element bildet die Transformationseinheit, die dynamisch bei Updates der Liste übertragen wird.

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| id (S) | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine div-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| styleref (S) | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| class (S) | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| state (E) | (Optional) Der initiale Status des Widgets; state kann dabei die folgenden Werte annehmen:<br>initial = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „required“ noch „erroneous“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>required = Darstellung als MUSS-Element<br>erroneous = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>disabled = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>readonly = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist state="initial". |
| tabindex (I) | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem tabindex ist fokussierbar, also auch ein Label; das Weglassen des tabindex-Attributs hat automatisch zur Folge, dass |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible="`true`") oder nicht (visible="`false`"); Standard-Einstellung ist visible="`true`". |
| `rolocked (B)` | (Optional) Wenn „true", verhindert Hittesting und die Erkennung von Events im „readonly"-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly"-Zustand auf „mouse-over"-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true", erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock"-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset") um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>"-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `extendable (B)` | (Optional) Falls „true", kann der Anwender zusätzliche Elemente (Zeilen) innerhalb des Datagrids anlegen; Default ist „false". |
| `autosize (E)` | (Optional) Definiert den Anzeige-Modus des Elements bei Größenänderung; mögliche Werte für dieses Attrubut sind:<br>„none" = keine Anpassung<br>„cols" = Spaltenbreiten werden angepaßt<br>„rows" = Zeilenhöhen werden angepaßt<br>„all" = Spaltenbreiten und Zeilenhöhen werden angepasst |
| `header (E)` | (Optional) Legt fest, ob die Spalten und/oder Zeilen einen Header besitzen; mögliche Werte sind somit: „none", „cols", „rows", „all" |
| `selmode (E)` | (Optional) Legt fest, was bzw. wie der Anwender einzelne Elemente des Datagrids selektieren kann; mögliche Werte sind: „CellSelect", „FullRowSelect", „FullColumnSelect", „RowHeaderSelect" und „ColumnHeaderSelect" |
| `multisel (B)` | (Optional) Falls „true", kann mehr als eine Zelle, Zele oder Spalte selektiert werden; Default ist „false". |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerte- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | ten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`editCell( cell )`**

Versetzt die Zelle „cell“ in den Bearbeitungsmodus.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**selectRows( from, to)**

Selektiert die Zeilen beginnend mit „from“ bis einschließlich „to“.

**selectCols( from, to)**

Selektiert die Spalten beginnend mit „from“ bis einschließlich „to“.

**selectCells( from, to)**

Selektiert die Zellen beginnend mit „from“ bis einschließlich „to“.

**insertRows( rows, idx)**

Fügt die Zeilen „rows“ hinter der Zeile „idx“ ein.

**removeRows( from, to)**

Löscht die Zeilen beginnend mit „from“ bis einschließlich „to“.

**clearSelection()**

Löscht jegliche Auswahl (Selection) innerhalb des Datagrids.

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**beginedit [ cellid ]**

Getriggert, nachdem eine Zelle in den Edit-Zustand versetzt worden ist.

**endedit [ cellid ]**

Getriggert, nachdem eine Zelle den Edit-Zustand verlassen hat.

**cellclicked [ cellid ]**

Getriggert, nachdem eine zelle angeklicked worden ist.

**cellcontentclicked [ cellid ]**

Getriggert, nachdem der Inhalt einer Zelle angeklicked worden ist.

**celldoubleclicked []**

Getriggert, nachdem der Inhalt einer Zelle doppelt angeklicked worden ist.

**headerclicked [ colid/rowid ]**

Getriggert, nachdem der Header einer Spalte oder Zeile angeklicked worden ist.

**rowclicked [ rowid ]**

Getriggert, nachdem der eine Zeile angeklicked worden ist.

**rowadded [ rowid ]**

Getriggert, nachdem der Anwender eine Zeile hinzugefügt hat.

**selectionchanged [ id ]**

Getriggert, nachdem sich die Auswahl durch den Anwender geändert hat.

### Beispieldarstellung

```
<bw:datagrid id="CP_KVGrid" tabindex="18" extendable="false">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
            eox="10" eoy="10" ecx="500" ecy="500"/>
  <bw:content>
    <bw:cols>
      <bw:col width="20" id="c1" type="image"></bw:col>
      <bw:col width="20" id="c2" type="checkbox"></bw:col>
      <bw:col width="90" id="c3" type="textbox">
        <bw:t>Bereich</bw:t>
      </bw:col>
      <bw:col width="65" id="c4" type="combobox">
        <bw:t>Tarif</bw:t>
      </bw:col>
    </bw:cols>
    <bw:rows>
      <bw:foreach element="Objekt.Liste.*Element">
        <bw:row id="{pos()}">
          <bw:cell id="{.EinschlussArt}">images/angebot.gif</bw:cell>
          <bw:cell readonly="{.Ausgewaehlt.ReadOnly}">
            <bw:var expr=".Ausgewaehlt"/>
          </bw:cell>
          <bw:cell styleref="labelStyle" readonly="true">
            <bw:var expr=".Info.Schlagw"/>
          </bw:cell>
          <bw:cell>
            <bw:var expr=".Tarif"/>
          </bw:cell>
        </bw:row>
      </bw:foreach>
    </bw:rows>
  </bw:content>
  <bw:events>
    <bw:event method="doThis" name="validate"/>
  </bw:events>
</bw:datagrid>
```

Die folgende Abbildung zeigt ein typisches Datagrid.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

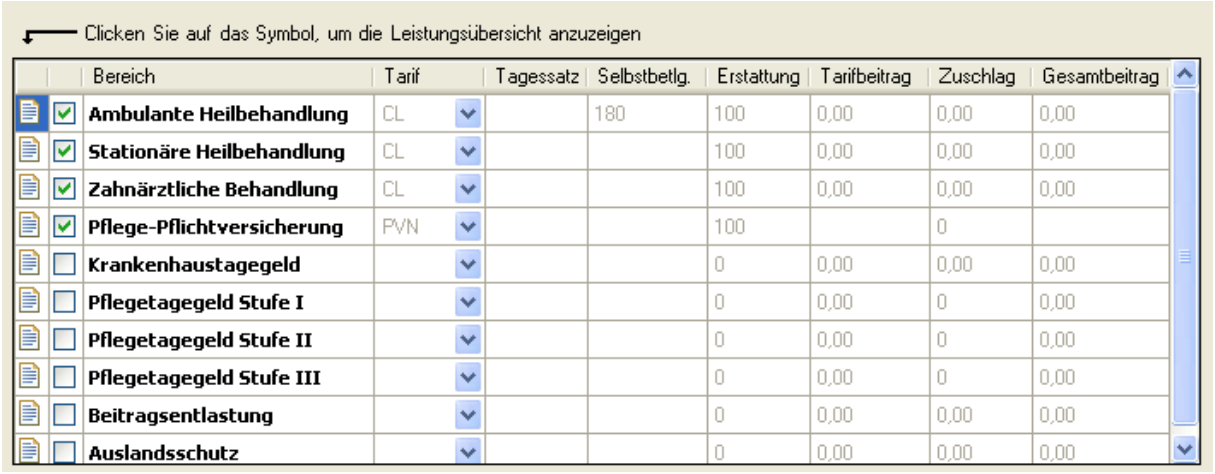



**Abb. 29: Beispiel eines Datagrid**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.15 <bw:c_list>, <bw:h_list>

### Kurzbeschreibung

Eine `<bw:c_list>` ist eine Grid-ähnliche Liste, welche ihre Daten in einer 2D-Matrix anordnet (ähnlich <bw:grid>) jedoch über weiter führende Styling-Möglichkeiten verfügt. Die `<bw:c_list>` ist in DHTML (also JavaScript/CSS und als BML-Rule) implementiert und bietet damit noch weiter gehende Möglichkeiten bezüglich Konfiguration, individuelle Anpassung und Layout.

Eine `<bw:h_list>` bietet dieselben Möglichkeiten wie die `<bw:c_list>` allerding innerhalb eines `<bw:browser>` Controls. Die `<bw:c_list>` erzeugt innerhalb der zugehörigen Rule ihr eigenes `<bw:browser>`-Control und ist für den Einsatz jeweils einer Liste inenrhalb eines Dialogs gedacht. Sollen mehrere Listen innerhalb eines Dialogs erzeugt werden, sollte aus Performance- und Speicherplatz-Gründen die `<bw:h_list>` verwendet werden. Bei Nutzung der `<bw:h_list>` innerhalb eines `<bw:browser>`-Controls müssen allerdings bestimmte Vorbedingungen beachtet werden, insbesondere müssen bestimmte JavaScript- und CSS-Dateien inkludiert werden (siehe unten).

Da beide Listen mit Hilfe einer BML-Rule in das benötigte Format konvertiert werden, muss die zugehörige Rule-Datei inkludiert werden:

```
<bw:include src="System_BWTable.inc" />
```

Zudem können individuelle Styling-Informationen innerhalb einer CSS-Datei sowie ggf. erforderliche Skripte innerhalb einer JavaScript-Datei definiert warden. Diese werden mit Hilfe des folgenden Konstrukts bei der `<bw:c_list>` verwendet:

```
<bw:var name="javascript_file" expr="'scripts/Test_DataTable.js'" />
<bw:var name="css_file" expr="'styles/Test_DataTable.css'" />
```

Da bei Verwendung der `<bw:h_list>` der entsprechende XHTML-Header explizit definiert warden muss, können die gewünschten CSS- bzw. JavScript-Dateien darin definiert bzw. inkludiert werden.

ID-Konvention: „EX_CL_“. (“EX_” für “Extended Controls”)

### Syntax (identisch für `<bw:h_list>`)

```
<bw:c_list
      id="MyList"
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      ieversion="9"
      tabindex="1"
      info="true"
      hrow="true"
      paging="false"
      pinfotext="Seite 1 von 3"
      coltoggle="true"
      filter="false"
      seltype="multiple"
      showtips="true"
      showexpwait="true"
      expandable="true"
      emptymsg="Keine Daten verfügbar"
      hovermsg="Alle anzeigen|Zeile anzeigen|
                Alle verstecken|Zeile verstecken"
      selrowsbyidx="0,2,3"
      selrowsbyid="R_0,R_1,R_4">

      <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
            eox="0" eoy="0" ecx="0" ecy="0" />

      <bw:cols>
            <bw:col name="Name" width="200" sortable="true"
                  visible="false" resizable="true"
                  allowtoggle="false" />
      </bw:cols>

      <bw:titles>
            <bw:title>
                  <bw:val>Spaltenüberschrift 1</bw:val>
            </bw_title>
            <bw:title>
                  <bw:val>Spaltenüberschrift 1</bw:val>
                  <bw:tip>Optionaler Titeltooltip</bw:tip>
            </bw_title>
      </bw:titles>

      <bw:data>
            <bw:foreach element=DOM.Path">
                  <bw:row>
                        <bw:cell>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
                              <bw:val>Dateninhalt Spalte 1</bw:val>
                        </bw:cell>
                        <bw:cell>
                              <bw:val>Dateninhalt Spalte 2</bw:val>
                              <bw:tip>Optionaler Zellentooltip</bw:tip>
                        </bw:cell>
                  </bw:row>
                  <bw:exprow>Optionale Zeilenerweiterung, die vom
                        Anwender mit Hilfe der +/- Tasten angezeigt
                        oder wieder versteckt werden kann.
                  </bw:exprow>
            </bw:foreach>
      </bw:data>

      <bw:events>
            <bw:event name="selectionchanged" method="" />
            <bw:event name="validate"  method="" />
            <bw:event name="expandrow" method="" />
            <bw:event name="layoutchanged" method="" />
      </bw:events>
</bw:c_list>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle). Die Tabelle beschreibt sowohl die Attribute des Elements `<bw:c_list>` (bzw. `<bw:h_list>`) als auch alle erlaubten inneren Elemente sowie deren mögliche Attribute.

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `ieversion (I)` | (Optional) Definiert die zu verwendende Version des Internet Explorers; da immer der auf dem Zielsystem installierte Internet Explorer verwendet wird, kann hier nur ein Hinweis auf die Kompatibilitätsstufe gegeben werden; die `<bw:c_list>` und die `<bw:h_list>` benötigen jedoch mindestens die IE-Version 8, um korrekt zu funktionieren; Standard-Wert ist IE=8. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist; hat die Liste den Fokus bekommen, springt dieser Innerhalb der Liste zwischen dem Menü-Button, dem Filter-Eingabefeld sowie der ersten Datenzeile, bis er die Liste wieder verlässt. |
| `info (B)` | (Optional) Stellt innerhalb der Liste den Info-Bereich (links unten) dar (info="`true`") oder nicht (info="`false`"); Standard-Einstellung ist info="`true`"; der Server kann hier mit Hilfe der API-Funktion „setInfo()" beliebige Text- bzw. XHTML-Inhalte anzeigen; es sollte bedacht werden, dass dieser bereich normalerweise eine Höhe von ca. 24 Pixeln hat und daher nur einzeilige Texte zulässt. |
| `hrow (B)` | (Optional) Falls „true" definiert beim Mouse-Over über einer Zeile eine CSS-Klasse, die entsprechend den Anforderungen gestyled werden kann; Standard ist hrow="false"; bezüglich der CSS-Definition siehe die Datei „bw_table_userstyling.css" |
| `paging (B)` | (Optional) Falls „true" zeigt rechts unten die Bedienelemente zum Blättern innerhalb der Liste („Blätter-Buttons"); Standard ist paging="true". |
| `pinfotext (S)` | (Optional) Ein String, der den Inhalt der Anzeige zwischen den Blätter-Buttons definiert; dieser Inhalt wird im Normalfall bei jedem erneuten Setzen des Listeninhalt mit übertragen und neu gesetzt, um z.B. die angezeigte Seite sowie ggf. die maximale Anzahl anzuzeigender Seiten zu präsentieren; wird kein Inhalt gesetzt, verwendet die Liste den Text „Seite 1 von 1". |
| `coltoggle (B)` | (Optional) Falls „true" zeigt links oben einen Button an, mit dem ein Menü angezeigt wird, das dem Anwender das ein- bzw. ausblenden einzelner Spalten erlaubt; Standard ist coltoggle="true"; sollen Spalten von der „Toggle"-Möglichkeit ausgenommen werden, muss für die entsprechende Spalte innerhalb des <bw:col>-Elements das Attribut „allowtoggle" auf den Wert ‚false' gesetzt werden (siehe dort). |
| `filter (B)` | (Optional) Wenn „true", zeigt rechts oben ein Eingabefeld für die Eingabe von Filterwerten ein; die Semantik des Filterwerts ist vom Server bzw. vom Prozess abhängig; Standardwert für filter ist „true"; der vom Anwender eingegebene Wert wird nach Drücken der ‚Enter'-Taste an den Server übertragen und löst normalerweise einen Reload der Listendaten mit dem gefilterten Inhalt aus. |
| `seltype (E)` | (Optional) Definiert den Typ der Selektion von Zeilen; mögliche Werte sind: „none" = es gibt keine Möglichkeit, eine Zeile zu selektieren bzw. als markiert darzustellen (in diesem Fall haben die Pfeiltasten eine leicht andere Funktion), „single" = es kann immer nur eine einzige Zeile selektiert/markiert werden (in diesem Fall verschieben die vertikalen Pfeiltasten die Selektion jeweils um eine Zeile) und |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | „multiple“ = es können beliebig viele Zeile selektiert bzw. markiert werden (Pfeiltasten-Verhalten analog dem Wert „none“);<br>Standard ist „none“. Hinweis: Diese Einstellung hat auch Auswirkungen auf die initiale Selektion von Zeilen (siehe unten). |
| `showtips (B)` | (Optional) Falls „true“ werden ggf. definierte Zusatzinformationen (sog. Tooltips) für einzelne Datenzellen angezeigt, anderfalls nicht; Standard ist „true“. |
| `showexpwait (S)` | (Optional) Falls 'true' (oder weggelassen) wird ein Marquee (Progressbar) innerhalb einer expandierten Zeile (Extra-Bereich) angezeigt, während die Daten vom Server geladen werden; Standard ist „true“; der Wert ‚false‘ macht vor allem dann Sinn, wenn das Laden der Zusatzdaten sehr schnell geht und eine intermediäre Anzeige stören würde. |
| `expandable (B)` | (Optional) Falls „true“ wird zu jeder Zeile (auch zu der Titelzeile) ein kleines Symbol hinzugefügt, mit dessen Hilfe der Anwender einen zusätzlichen Raum unterhalb der Zeile „öffnen“ kann, um darin weitere Informationen darzustellen (die sog. Row-Expansion); das „Expandieren“ dieses Raums triggert eine Message an den Server (siehe API), welche als Rückantwort ein XHTML-Fragment mit den zustzlich anzuzeigenden Daten liefern muss. Standard ist „true“. Das Symbol kann mit Hilfe der CSS-Datei „bw_table_userstyling.css“ definiert werden. |
| `emptymsg (S)` | (Optional) Definiert einen Text, der angezeigt wird, wenn die Liste leer ist; da es einen Standard-Text gibt, sollte ein leerer Text angegeben werden, wenn kein Text gewünscht wird. |
| `hovermsg (S)` | (Optional) Definiert – durch das Pipe-Symbol (\|) getrennt – insgesamt vier Texte, die als Tooltip für die Zeilen-Expansion-Symbole (normalerweise +/-) angezeigt werden; der erste Text wird für das Titelzeilen-Symbol „Alle anzeigen“ verwendet, der zweite Text für jede Datenzeile „Zeile anzeigen“, der dritte Text für das Titelzeilen-Symbol „Alle verstecken“ und der vierte Text für jede Datenzeile „Zeile verstecken“. |
| `selrowsbyid (S)` | (Optional) Definiert eine Komma-separierte Liste von Zeilen-IDs, die initial als markiert angezeigt werden sollen; dies aber nur dann, wenn der Wert von selmode auf „multiple“ steht; ist der Wert von selmode=“single“, wird nur die erste angegebene Zeile markiert, ist selmode=“none“ oder nicht definiert, wird keine Zeile markiert; Beispiel: „R_1,R_2,R_3“; Hinweis: Es dürfen keine Leerzeichen verwendet werden! |
| `selrowsbyidx (S)` | (Optional) Definiert eine Komma-separierte Liste von Zeilen-Indexwerten, die initial als markiert angezeigt werden sollen; dies aber nur dann, wenn der Wert von selmode auf „multiple“ steht; ist der Wert von selmode=“single“, wird nur die erste angegebene Zeile markiert, ist selmode=“none“ oder nicht definiert, wird keine Zeile markiert; Beispiel: „1,4,7,8“; Hinweis: Es dürfen keine Leerzeichen verwendet werden. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `<bw:cols>` | Definiert als Child-Elemente eine Anzahl von Spalten-Elementen <bw:col> (siehe diese) zusammen mit einer Reihe von Attributen zu jeder Spalte. |
| `<bw:col>` | Definiert ein Spalten-Element (also eine einzelne Spalte innerhalb der Tabelle) sowie einige Attribute, die das Verhalten der Spalte bestimmen; die Attribute sin in der folge angegeben: |
| `name (S)` | (Optional) Name der Spalte; dieser Name wird auch innerhalb des „Spalten ein-/ausblenden-Menüs“ verwendet; wird kein Name angegeben, verwendet die Liste der Inhalt des Titels dieser Spalte; die Angabe eines Namens verbessert die Performance beim initialen Aufbau der Liste |
| `width (S)` | (Optional) Definiert die Breite der Spalte in Pixel (die Angabe der Maßeinhalt kann in der Version 1.0 weggelassen werden, da aktuell nur Pixel unterstützt werden); alternativ zum Weglassen des width-Attributs kann auch der Wert „auto“ angegeben werden (entspricht dem Weglassen des width-Attributs); die Liste verteilt in diesem Fall die restliche (also die Breite der Tabell minus der Breite aller Spalten, die nicht als „auto“ definiert worden sind) initiale (!) Breite der Tabelle auf alle als „auto“ definierten Spalten; in jedem Fall wird jedoch eine Mindestbreite von 24 Pixel eingehalten. |
| `sortable (B)` | (Optional) Falls ‚true‘ kann der Anwender den Inhalt der Liste mit Hilfe dieser Spalte auf- bzw. absteigend sortieren; Standardwert ist ‚false‘ |
| `visible (B)` | (Optional) Falls ‚false‘ wird die Spalte initial unsichtbar dargestellt (also gar nicht angezeigt), der Anwender kann die Spalte jedoch mit Hilfe des Menüs (sofern vorhanden) wieder einblenden; Standardwert ist ‚true‘ |
| `resizable (B)` | (Optional) Falls ‚true‘ kann der Anwender die Breite der Spalte individuell ändern; Standardwert ist ‚false‘ |
| `allowtoggle (B)` | (Optional) Falls ‚false‘ wird die Spalte nicht innerhalb des Menüs zum Ein-/Ausblenden von Spalten angezeigt; dadurch kann der Anwender die Spalte nicht ausblenden (wurde sie initial als unsichtbar definiert aber auch nicht einblenden) |
| `<bw:titles>` | Wrapper-Element für die Definition von Spalten-Titeln |
| `<bw:title>` | Definiert den Inhalt der Titelzeile für die entsprechende Spalte; es müssen exakt die gleiche Anzahl von <bw:col>, <bw:title> und <bw:cell> Elementen in einer Tabelle vorkommen! |
| `<bw:val>` | Enthält den eigentlichen Inhalt der Spalte; bei diesem kann es sich um einen simplen Text oder um ein XHTML-Fragment (z.B. ein <img>-Element) handeln |
| `class (S)` | (Optional) Das <bw:val>-Element kann optional eine CSS-Class-Angabe beinhalten; diese Klasse wird dann dem entsprechenden -Element innerhalb der Tabelle hinzugefügt und ermöglicht das Styling der Titel-Inhalte (z.B. Farbe, Hintergrund, Abstand etc.) |
| `<bw:tip>` | (Optional) Jede Spaltenüberschrift (Titel-Zelle) kann optional ein Tooltip-Element enthalten, das insbesondere bei sehr schmalen Spalten die Semantik der Spalte erläutern kann; auch hier kann entweder ein einfacher Text als Inhalt oder ein XHTML-Fragment verwendet werden. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `<bw:data>` | Enthält die Defintion der Daten (also Zeilen bzw. Zellen); das <bw:data>-Element dient als „Wrapper“ um die eigentlichen Zeilen- bzw. Spaltendefinitionen (siehe unten) |
| `<bw:row>` | Definiert eine Zeile, bestehend als N Zellen; kann optional ein vordefiniertes „Expansion“-Element enthalten (<bw:exprow>, siehe unten), das vom Anwender ein- bzw. ausgeschaltet werden kann |
| `<bw:cell>` | Definiert eine Datenzelle, also innerhalb einer Zeile den jeweiligen Spalteninhalt; es müssen exakt gleich viele <bw:col>, <bw:title> und <bw:cell> Elemente innerhalb einer tabelle vorhanden sein! |
| `<bw:val>` | Enthält den eigentlichen Inhalt der Spalte (Zelle); bei diesem kann es sich um einen simplen Text oder um ein XHTML-Fragment (z.B. ein <img>-Element) handeln |
| `class (S)` | (Optional) Das <bw:val>-Element kann optional eine CSS-Class-Angabe beinhalten; diese Klasse wird dann dem entsprechenden -Element innerhalb der Tabelle hinzugefügt und ermöglicht das Styling der Titel-Inhalte (z.B. Farbe, Hintergrund, Abstand etc.) |
| `<bw:tip>` | (Optional) Jede Zelle kann optional ein Tooltip-Element enthalten, das zusätzliche Informationen zu der Zelle anzeigen kann (wenn der Anwender mit der Maus darüber fährt); auch hier kann entweder ein einfacher Text als Inhalt oder ein XHTML-Fragment verwendet werden |
| `<bw:events>` | (Optional) Definiert gewünschte Callbacks an den Server (Wrapper für entsprechende <bw:event> Elemente, siehe unten) |
| `<bw:event>` | (Optional) Definiert ein Callback für ein bestimmtes Event; die verfügbaren Events mitsamt der Argumente-Beschreibung siehe unten (API-Funktionen) |
| `name (S)` | Name des Events |
| `method (S)` | Gewünschte Callback-Angabe mit Argumenten |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

Hinweis: Jede API-Funktion hat grundsätzlich maximal zwei Argmente: „data“ und „jopts“. „data“ ist entweder ein String oder ein XHTML-Fragment, jopts eine „stringified“ JSON-Struktur mit einigen Attributen (siehe unten). „data“ dient im Normalfall der Übermittlung der eigentlich Argument-Daten, „jopts“ der Übermittlung zusätzlicher Informationen über btw. an die Liste.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setListData ( data, jopts )`**

Setzt den Inhalt der Liste auf den als Argument `data` übergebenen Wert. Bei „data“ muss es sich um ein XHTML-Fragment mit folgender Struktur handeln:

```
<tbody bw_pginfo="Seite 2 von 13">
  <tr>
       <td>
            <div class="cv">Data for column 1</div>
       </td>
       <td>
            <div class="cv">Data for column 2</div>
       </td>
       ...
  <tr>
</tbody>
```

Dieses Fragment kann direkt vom Server erzeugt als Ergebnis einer (Teil-) Transformation der entsprechenden BML-Seite erzeugt werden. Das Attribut „bw_pginfo“ beinhaltet den Text der Blätterbuttons; der Rest entspricht dem einfachen Tabellenaufbau einer XHTML-Tabelle; wird das Fragment vom Server native erzeugt, muss bedacht werden, dass die -Elemente innerhalb der Zellen zusätzliche Klassen-Angaben enthalten können. Bei der Native-Erzeugung muss auch bedacht werden, dass ggf. die „Expansion“-Spalte sowie die „Filler“-Spalte mit definiert werden müssen (siehe die entsprechende Rule-Datei). Das Argument „jopts“ ist die oben erwähnte JSON-Struktur, die beim Callback vom Client an den Server mit übertragen wird und lediglich wieder zum Client „durchgeschleift“ werden muss. Wird dieses API jedoch nicht als Callback aufgerufen, muss der Server diese Struktur selbst erzeugen. Die „jopts“-Struktur sieht wie folgt aus (Javascript-Notation):

```
var jOpts = {
 "data": {
       "cmd": null,
       "targetid": null,
       "reserve": null
 },
 "success": "bw_successCallback",
 "error": "bw_errorCallback",
 "reserve": null
};
```

„cmd“ enthält das gewünschte Command (siehe diese Beschreibung der API-Funktionen) als String (z.B. „setListData“), „targetid“ die ID der Liste (wie im ID-Attribut angegeben),


© Copyright BISS GmbH, 2001-2012 All Rights reserved

„success“ und „error“ die entsprechenden Entry-Points für die Javascript-Funktionen (diese beiden Werte dürfe nicht geändert werden); alle anderen Werte sind von der gewünschten API-Funktion abhängig. „setListData“ verwendet keine weiteren Werte innerhalb der „jopts“-Struktur.

**`appendListData ( data, jopts )`**

Fügt die Daten „data“ (XHTML-Fragment, siehe oben) ans Ende der aktuellen Listendaten an. Die Struktur der Daten sowie von „jopts“ entspricht dem API „setListData“.

**`removeListData ( data, jopts )`**

Löscht alle Daten aus der Tabelle. Das Layout der Tabelle (also z.B. Spaltenbreiten, Spaltensichtbarkeiten etc.) bleiben unverändert. Nach Löschen der Daten wird der Text „data“ angezeigt. Dabei kann es sich um einen einfachen Text oder ein passendes XHTML-Fragment handeln. „removeListData“ verwendet keine weiteren Werte innerhalb der „jopts“-Struktur.

**`setInfo ( data, jopts )`**

Setzt den Infobereich der Tabelle auf den Wert „data“. Dabei kann es sich um einen einfachen Text oder ein XHTML-Fragment handeln. „setInfo“ verwendet keine weiteren Werte innerhalb der „jopts“-Struktur.

**`setFilter ( data, jopts )`**

Setzt den Wert des Filter-Eingabefeldes auf den Wert „data“. Dabei muss es sich um einen einfachen Text-String handeln. Ist das Argument „trigger“ = ‚true‘ wird unmittelbar der entsprechende Callback getriggert. „trigger“ wird im Feld „data.reserve“ der „jopts“-Struktur übertragen.

**`setFragment ( data, jopts )`**

Setzt den Inhalt der expandierten Zeile. Die Zeile wird mit Hilfe des ID-Attributs angegeben, das im Feld „data.reserve“ der „jopts“-Struktur übertragen wird. Beim „Ziel“ (also dem „Container“ innerhalb der expandierten Zeile) handelt es sich um ein -Element. Beim Argument „data“ handelt es sich um ein XHTML-Fragment.

**`setItemsSelected ( data, jopts )`**

Setzt die in „data“ definierten Zeilen in den Zustand „selektiert“ bzw. „markiert“. Dies entspricht einer initialen Definition des Attributs „selrowsbyid“. Aktuell können für „data“ ausschließlich IDs verwendet werden. „setItemsSelected“ verwendet keine weiteren Werte innerhalb der „jopts“-Struktur.

**`enable ()`**

Versetzt die gesamte Liste in den Zustand „enabled“, d.h. aktiv und bedienbar. Die Funktion verwendet keine Argumente.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`disable ()`**

Versetzt die gesamte Liste in den Zustand „disabled“, d.h. inaktiv und nicht bedienbar. Die Funktion verwendet keine Argumente.

**`enablePaging ()`**

Versetzt die Blätter-Buttons in den Zustand „enabled“, d.h. aktiv und bedienbar. Die Funktion verwendet keine Argumente.

**`disablePaging ( data )`**

Versetzt die Blätter-Buttons in den Zustand „disabled“, d.h. inaktiv und nicht bedienbar. Der Paging-Text bleibt dabei normal sichtbar, lediglich die Buttons werden normalerweise abgegraut und inaktiv dargestellt. Dieser Zustand sollte vom Server gesetzt werden, wenn die angezeigte Datenseite die einzige Seite ist und der Anwender nicht blättern kann, der Server sollte zudem den passenden Text setzen (optionales Argument „data“).

## Events

**`requestdata [ @listid, @filterval, @sortcol, @sortdir, @event ]`**

Getriggert, wenn der Benutzer indirekt (z.B. durch Klick auf einen der Blätterbuttons) eine neue „Seite“ mit Daten anfordert. Da dies durch drei verschiedene Benutzeraktionen erfolgen kann, wird die auslösende Aktion im Argument @event mitgegeben. Das Event liefert die Argumente @listid (ID der Liste, wie im ID-Attribut angegeben), @sortdir („asc“ für aufsteigende Sortierung bzw. „desc“ für absteigende Sortierung), @sortcol (Name der Spalte wie im Attribut „name“ angegeben oder – falls kein „name“-Attribut verwendet wurde, den Inhalt der Spaltenüberschrift als Text), @filterval (Inhalt des Filter-Eingabefeldes) sowie @event („filter“, falls das Event durch Drücken der ‚Enter‘-Taste innerhalb des Filter-Eingabefeldes ausgelöst wurde, „sort“, falls das Event durch die Sortierung nach einer Spalte ausgelöst wurde, „pg_first“, „pg_prev“, „pg_next“ oder „pg_last“, falls das Event durch Klick auf einen der Blätter-Buttons ausgelöst wurde).

**`selectionchanged [ @listid, @rowid, @selected ]`**

Getriggert, wenn der Benutzer die Zeilenauswahl ändert (z.B. durch Anklicken einer bestimmten Zeile). Das Event liefert die Argumente @listid (ID der Liste, wie im ID-Attribut angegeben), @rowid (ID der Zeile, wie im ID-Attrinut angegeben oder den 0-basierten Index der Zeile falls keine ID angegeben wurde) sowie @selected (‚true‘ falls die Zeile selektiert bzw. ‚false‘ falls die Zeile deselektiert worden ist.

**`validate [ @listid, @rowid ]`**

Getriggert, wenn Anwender eine zeile selektiert und die ‚Enter‘-Taste drückt. Das „selmode“-Attribut der Liste muss dabei den Werte „single“ haben! Das Event liefert die Argumente


© Copyright BISS GmbH, 2001-2012 All Rights reserved

@listid (ID der Liste, wie im ID-Attribut angegeben) sowie @rowid (ID der Zeile, wie im ID-Attrinut angegeben oder den 0-basierten Index der Zeile falls keine ID angegeben wurde).

**`expandrow [ @rowid ]`**

Getriggert, wenn der Anwender eine Zeile mit Hilfe des + Symbols expandiert hat. Das Event liefert das Argument @rowid (ID der Zeile, wie im ID-Attrinut angegeben oder den 0-basierten Index der Zeile falls keine ID angegeben wurde). Hinweis: Wird die zusätzlich Info der Zeile (mit Hilfe des – Symbols) wieder versteckt, wird kein Event getriggert!

**`layoutchanged [ @listid, @event, @colname, @visible, @width ]`**

Getriggert, nachdem der Anwender das Layout der Tabelle geändert hat. Dies kann geschehen, wenn der Anwender eine Spalte sichtbar oder unsichtbar macht oder die Breite einer Spalte ändert. Das Event liefert die folgenden Argumente: @listid (ID der Liste, wie im ID-Attribut angegeben), @event (entweder „toggle“, falls eine Spalte sichtbar/unsichtbar gemacht wurde, oder „width“, falls die Breite einer Spalte geändert wurde), @colname (Name der Spalte wie im Attribut „name“ angegeben oder – falls kein „name“-Attribut verwendet wurde, den Inhalt der Spaltenüberschrift als Text), @visible (‚true‘ falls die Spalte sichtbar ist, ‚false‘, falls sie unsichtbar ist) sowie @width (Breite der Spalte in Pixel).

## Beispieldarstellung

```
<bw:c_list id="CL_TestListe2"
   ieversion="8"
   tabindex="2"
   info="true"
   hrow="true"
   emptymsg="Bitte klicken Sie auf Suchen, um Daten in der Liste anzuzeigen">

   <bw:pos aox="0" aoy="500" acx="1000" acy="1000"
           eox="0" eoy="10" ecx="0" ecy="0" />

   <bw:cols>
      <bw:col name="VSNR" width="180" resizable="true" />
      <bw:col name="Sparte" width="200" resizable="true" />
      <bw:col name="Deckung" width="220" resizable="true" />
      <bw:col name="Beitrag" width="120" resizable="false" />
      <bw:col name="Schäden" width="100" resizable="true" />
   </bw:cols>

   <bw:titles>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      <bw:title><bw:val>VSNR</bw:val></bw:title>
      <bw:title><bw:val>Sparte</bw:val></bw:title>
      <bw:title><bw:val>Deckung/VP</bw:val></bw:title>
      <bw:title><bw:val>Beitrag</bw:val></bw:title>
      <bw:title><bw:val>Schäden</bw:val></bw:title>
   </bw:titles>

   <bw:data>
      <bw:foreach element="[0..9]">
         <bw:row id="R_{pos()+1}">
            <bw:cell><bw:val>430.222.12.111</bw:val></bw:cell>
            <bw:cell><bw:val>Leben</bw:val></bw:cell>
            <bw:cell><bw:val>Max Mustermann</bw:val></bw:cell>
            <bw:cell><bw:val>1.230,40 EUR</bw:val></bw:cell>
            <bw:cell><bw:val>Keine</bw:val></bw:cell>
         </bw:row>
      </bw:foreach>
   </bw:data>

   <bw:events>
      <bw:event name="selectionchanged"
         method="handleEvent( 'SEL_CHG', @listid, @rowid, @selected)" />
      <bw:event name="validate"
         method="handleEvent( 'ROW_VALIDATE', @listid, @rowid)" />
      <bw:event name="expandrow"
         method="requestDOM( TestFragment.bml)" />
   </bw:events>

</bw:c_list>
```

Die folgende Abbildung zeigt eine `<bw:c_list>` mit Beispieldaten.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

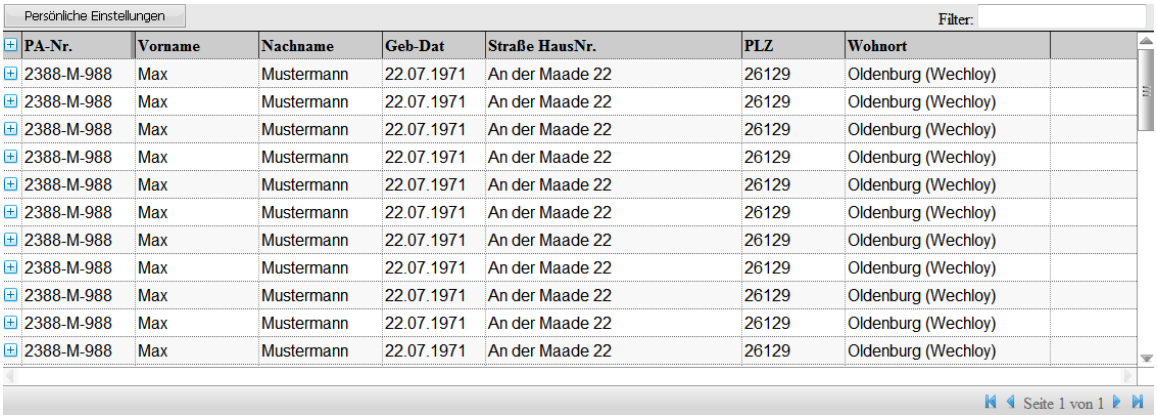

Persönliche Einstellungen
Filter:
PA-Nr. Vorname Nachname Geb-Dat Straße HausNr. PLZ Wohnort
2388-M-988 Max Mustermann 22.07.1971 An der Maade 22 26129 Oldenburg (Wechloy)
Seite 1 von 1

**Abb. 30: Beispiel für eine c_list**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.16 <bw:tree>

### Kurzbeschreibung

Ein `<bw:tree>`-Element stellt eine Baumansicht dar. Auf jeder Ebene des Baumes Texte kombiniert mit einem Icon angezeigt werden.

### Syntax

```
<bw:tree
  id="str_widget_id"
  styleref="str_style_reference"
  class="str_styleclass"
  tabindex="int_tabindex"
  state="enum_state"
  enabled="bool"
  visible="bool"
  startidx="int_index"
  checkstyle="bool"
  >
  <bw:content>
    <bw:tbody>
      <bw:ti id="item_id" expanded="bool">
        Root-Element (1. Ebene)
      </bw:ti>
      <bw:tdata element="list_path">
        <bw:ti id="item_id">
          Inhalt des Items in der 2. Ebene
        </bw:ti>
        <bw:tdata element="list_path">
          <bw:ti>
            Inhalt des Items in der 3. Ebene
          </bw:ti>
        </bw:tdata>
      </bw:tdata>
    </bw:tbody>
  </bw:content>
</bw:tree>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `checkstyle (B)` | (Optional) Falls „true“, zeigt zu Beginn (links) eines jeden Tree-Element eine Checkbox an; Default ist „false“. |
| `padding (I)` | (Optional) Definiert ein vertikales Padding in Pixel eines jeden Tree-Elements; je größer das padding, umso weiter voneinander werden die Elemente dargestellt. |
| `selidx (I)` | (Optional) 0-basierter Index der initual dargestellten Auswahl (Selection). |
| `selid (S)` | (Optional) Die ID des Tree-Elements für initiale Auswahl, bei nicht spezifiziertem „selidx“-Attribut |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`clear()`**

Löscht jegliche Auswahl (Selection).

**`removeItems( from, to)`**

Löscht die Elemente des Trees beginned mit „from“ bis einschließlich „to“.

**`setItemsChecked( from, to, state)`**

Setzt die Elemente beginnen mit „from“ bis einschließlich „to“ auf den Status „state“ (angehakt oder nicht angehakt).

**`setItemsExpanded( from, to, state)`**

Setzt die Elemente beginnen mit „from“ bis einschließlich „to“ auf den Status „state“ (expandiert oder nicht expandiert).

**`setItemsSelected()`**

Setzt die Elemente beginnen mit „from“ bis einschließlich „to“ auf den Status „state“ (selektiert oder nicht selektiert).

**`setItemsImage( id, img)`**

Setzt das Image des Items „id“ auf den Wert „img“.

**`setItemsStyle( id, style)`**

Setzt das Aussehen (den Style) des Items „id“ auf den Wert „style“.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**revealItems( from, to)**

Zeigt die Tree-Elemente beginnend mit „from“ bis einschließlich „to“.

**removeSubItems( id)**

Löscht alle Unterelemente des Elements mit der ID „id“.

**setSubItems( id, items)**

Setzt die Unterelemente des Items mit der ID „id“ auf den Wert „items“.

**addItems( items)**

Fügt „items“ ans Ende des Trees an.

**insertItems( items, id)**

Fügt „items“ hinter das Item mit der ID „id“ ein.

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**checkrequest []**

Getriggert, nachdem der Anwender eine entmarkierte (nicht angehakte) Checkbox markiert hat, aber bevor die Checkbox tatsächlich geändert wird.

**uncheckrequest []**

Getriggert, nachdem der Anwender eine markierte (angehakte) Checkbox entmarkiert hat, aber bevor die Checkbox tatsächlich geändert wird

**check [ nid ]**

Getriggert, nachdem der Anwender eine entmarkierte (nicht angehakte) Checkbox markiert hat.

**uncheck [ nid ]**

Getriggert, nachdem der Anwender eine markierte (angehakte) Checkbox entmarkiert hat.

**selected [ nid ]**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**deselected [ nid ]**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**expand [ nid ]**

Getriggert, nachdem der Anwender einen Tree-Konten expandieren möchte, bevor dies jedoch tatsächlich erfolgt ist.

**collapse [ nid ]**

Getriggert, nachdem der Anwender einen Tree-Konten zusammenfalten möchte, bevor dies jedoch tatsächlich erfolgt ist.

**expanded [ nid ]**

Getriggert, nachdem der Tree-Knoten expandiert worden ist.

**collapsed [ nid ]**

Getriggert, nachdem der Tree-Knoten zusammengefaltet worden ist.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`nodeclicked [ nid ]`**

Getriggert, nachdem der Tree-Knoten angeklickt worden ist.

**`nodedoubleclicked []`**

Getriggert, nachdem der Tree-Knoten doppelt angeklickt worden ist.

### Hinweise

Als Inhalte von `<bw:ti>`-Elementen sind Text bzw. alle Elemente zugelassen, die Text generieren (z.B. das `<bw:var>`-Element). Zusätzlich ist das das `<bw:img>`-Element erlaubt.

### Beispieldarstellung

Die folgende Abbildung enthält ein einfaches `<bw:tree>`-Widget mit bis zu 4 Ebenen.

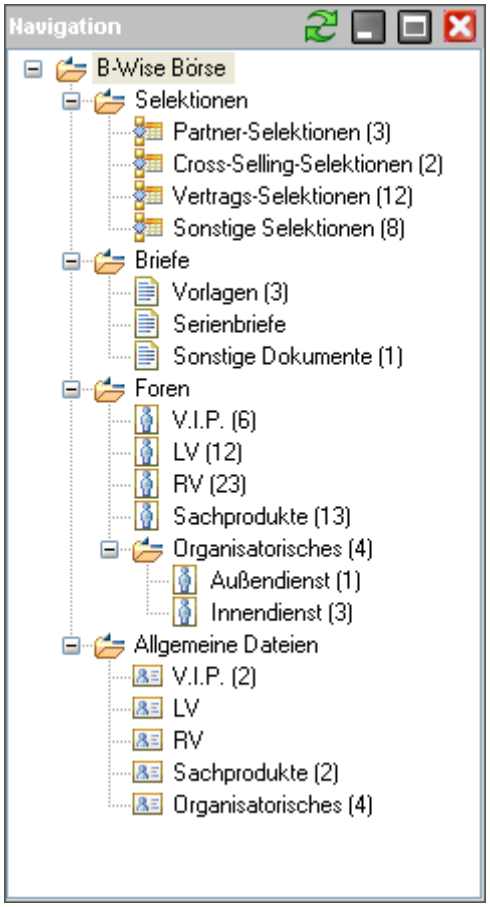



**Abb. 31: Beispiel eines Tree**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.17 <bw:portal>

### Kurzbeschreibung

Ein `<bw:portal>` ist ein spezielles Layout-Element zur Anordnung von Portlets auf einer Portalseite.

Eine Portalseite ist ein Fenster bzw. eine XHTML-Seite optional mit spezifischen Hilfselementen (z.B. Menubar, Toolbar, Statusarea) sowie einem <bw:portal>-Element, innerhalb dessen die einzelnen Portlets (siehe auch 9.5.13) platziert und angezeigt werden. Das grundsätzliche Layout zeigt schematisch die folgende Abbildung.

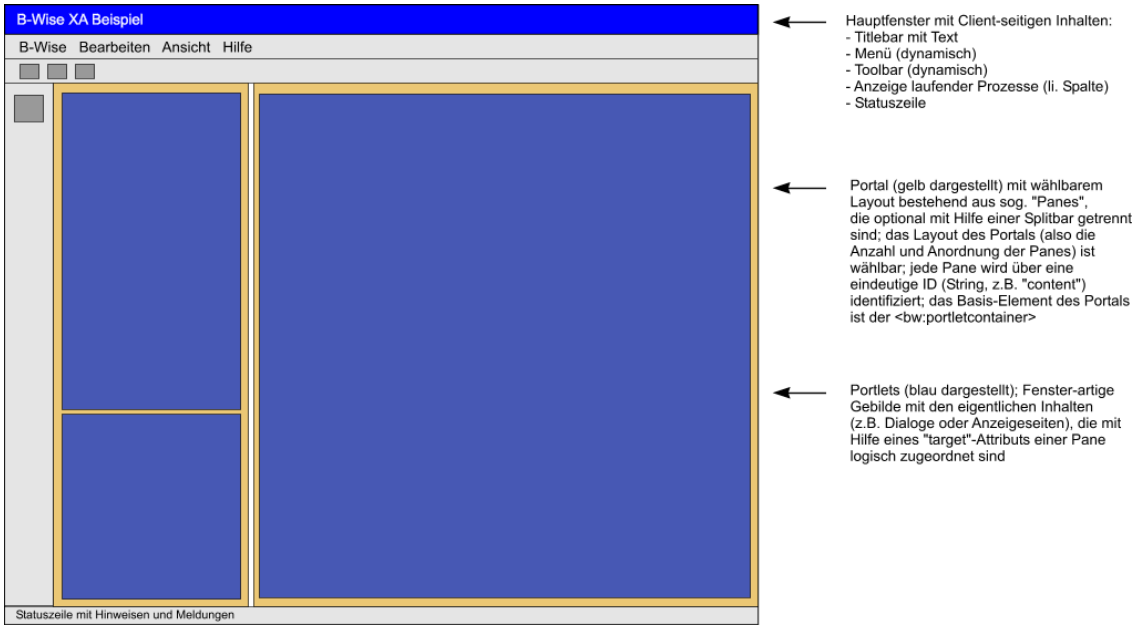

B-Wise XA Beispiel
B-Wise Bearbeiten Ansicht Hilfe
Statuszeile mit Hinweisen und Meldungen
Hauptfenster mit Client-seitigen Inhalten:
- Titlebar mit Text
- Menü (dynamisch)
- Toolbar (dynamisch)
- Anzeige laufender Prozesse (li. Spalte)
- Statuszeile
Portal (gelb dargestellt) mit wählbarem Layout bestehend aus sog. "Panes", die optional mit Hilfe einer Splitbar getrennt sind; das Layout des Portals (also die Anzahl und Anordnung der Panes) ist wählbar; jede Pane wird über eine eindeutige ID (String, z.B. "content") identifiziert; das Basis-Element des Portals ist der <bw:portletcontainer>
Portlets (blau dargestellt); Fenster-artige Gebilde mit den eigentlichen Inhalten (z.B. Dialoge oder Anzeigeseiten), die mit Hilfe eines "target"-Attributs einer Pane logisch zugeordnet sind

**Abb. 32: Layout-Schema eines *B-Wise* Fensters/einer Seite**

Innerhalb eines `<bw:portal>`-Elements können sowohl `<bw:portletcontainer>`- als auch `<bw:portletpane>`-Elemente enthalten sein. Sinnvollerweise hat ein `<bw:portal>`-Element mindestens ein `<bw:portletpane>`-Element, innerhalb dessen Portlets angezeigt werden können.

Innerhalb eines `<bw:portletcontainer>`-Elements können wiederum sowohl `<bw:portletcontainer>`- als auch `<bw:portletpane>`-Elemente inkludiert werden, so dass sich eine beliebig rekursive Struktur ergibt, die mächtige Layouts zulässt. Ein `<bw:portletpane>`-Element ist das „Blatt“ der Hierarchie, kann also keine weiteren Elemente aufnehmen.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Ein `<bw:portletpane>`-Element kann (über seine `id`) als Ziel (`target`) eines Portlets dienen. Ein Portlet, dessen `target`-Attribut also mit der `id` eines `<bw:portletpane>`-Elements übereinstimmt, wird innerhalb des Pane-Elements dargestellt. Die Darstellung folgt dabei bestimmten Regeln, die primär vom Portletcontainer vorgegeben werden; sie kann jedoch in gewissen Grenzen durch das Portlet selbst bestimmt werden (z.B. kann das Portlet eine Höhe vorgeben, die jedoch ggf. vom Portletcontainer ignoriert werden kann !).

Portal-Layouts werden jedes innerhalb einer eigenen Datei definiert, welche gemäß den B-Wise Konventionen die Dateiendung „.por“ trägt.

### Syntax

```
<bw:portal
  id="str_widget_id"
  defaulttarget="str_target"
  >
  <bw:portletcontainer
    id="str_widget_id"
    x="pos-x"
    y="pos-y"
    w="width"
    h="height"
    split="split-orientation"
    />
  <bw:portletpane
    id="widget-id"
    x="pos-x"
    y="pos-y"
    w="width"
    h="height"
    radio="radio-mode"
    />
</bw:portal>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine div-Section und muss daher auch eindeutig in Bezug auf alle anderen Ids der BML-Seite sein. Die ID einer Pane ist dabei die Ziel- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | adresse eines Portlets (siehe auch `target`-Attribut eines Portlets). |
| `defaulttarget (S)` | (Optional) Definiert eine ID (Name) für ein Default-Target innerhalb des Portals; diese wird verwendet, falls innerhalb der Unterelemente (Pane) kein target explizit definiert worden ist. |
| `split (E)` | (Optional) Definiert, ob der Container über eine horizontale (split="H") oder vertikale (split="V") Splitbar verfügen soll. Standardwert ist keine Splitbar. |

## API-Funktionen

**`removePortlets( portlet_ids )`**

Löscht alle Portlets, deren IDs als Argument übergeben worden sind aus dem Portal („explizites Löschen"). Wird der Aufruf ohne Argument verwendet (oder mit einem `null`-Argument), werden alle (non-sticky) Portlets aus dem Portal entfernt („implizites Löschen"). Beim expliziten Löschen (d.h. wenn IDs übergeben werden), wird auf dem Client eine Fehlermeldung angezeigt (MessageBox), falls eines der Portlets nicht gefunden werden kann.

**`addPortlets( portlet_ids )`**

Fügt die Portlets, deren IDs als Argument übergebenen werden, zum Portletcontainer (Portal) hinzu. Die neuen Portlets werden in den entsprechenden Panes angezeigt, die durch das `target`-Attribut innerhalb des jeweiligen Portlets definiert sind.

**`createDlg( dialog )`**

Erzeugt den als Argument übergebenen Dialog und zeigt ihn an.

**`execCmd( cmd )`**

Führt den als Argument übergebenen Befehl aus.

**`setBusy( txt )`**

Verhindert Benutzereingaben, in dem das Portal auf inaktiv gesetzt wird. Optional kann ein als Argument (`txt`) übergebener Text innerhalb des Statusbereichs des Portals angezeigt werden.

**`setIdle()`**

Gegenstück zu setBusy. Ein ggf. von setBusy im Statusbereich gesetzter Text wird entfernt.

**`setStatus( txt )`**

Zeigt den als Argument übergebenen Text im Statusbereich des Portals an.

**`setError( txt )`**

Zeigt den als Argument übergebenen Text im Statusbereich des Portals an (für Fehleranzeigen ist innerhalb des Statusbereich ein besonderer Raum reserviert).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**removePortal()**

Löscht das gesamte Portal inkl. aller Portlets. Es können keine weiteren Portlets hinzugefügt werden, bis ein neues Portal erzeugt worden ist.

**setPortal( portal )**

Definiert ein neues Portal, dessen Layout (XML-Repräsentation) als Argument übergeben worden ist.

**resetLayout()**

Setzt das Portal-Layout in den Ausgangszustand zurück (Position der Splitbars).

### Beispieldarstellung

```
<bw:portal id="mainportal">
  <bw:portletcontainer id="c1" px="0" py="0" dx="100%" dy="100%" split="V">
    <bw:portletpane id="navi" px="0" py="0" dx="200px" dy="100%" radio="1"/>
    <bw:portletpane id="cont" px="N" py="0" dx="*" dy="100%" radio="0"/>
  </bw:portletcontainer>
</bw:portal>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.17.1 <bw:portletcontainer>

### Kurzbeschreibung

Ein `<bw:portletcontainer>` ist ein spezielles Layout-Element zur Anordnung von Portlets innerhalb eines Portals.

Innerhalb eines `<bw:portletcontainer>`-Elements können wiederum sowohl `<bw:portletcontainer>`- als auch `<bw:portletpane>`-Elemente inkludiert werden, so dass sich eine beliebig rekursive Struktur ergibt, die mächte Layouts zulässt. Ein `<bw:portletpane>`-Element ist das „Blatt“ der Hierarchie, kann also keine weiteren Elemente aufnehmen.

### Syntax

```
<bw:portal
   id="str_widget_id"
   defaulttarget="str_target"
   >
   <bw:portletcontainer
      id="str_widget_id"
      px="pos-x"
      py="pos-y"
      dx="width"
      dy="height"
      split="split-orientation"
      />
   <bw:portletpane
      id="widget-id"
      px="pos-x"
      py="pos-y"
      dx="width"
      dy="height"
      radio="radio-mode"
      />
</bw:portal>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | mapped beim Browser auf die ID eine div-Section und muss daher auch eindeutig in Bezug auf alle anderen Ids der BML-Seite sein. Die ID einer Pane ist dabei die Zieladresse eines Portlets (siehe auch `target`-Attribut eines Portlets). |
| `px, py (S)` | Start-Koordinaten (X- bzw. Y-Position) des Containers bzw. der Pane. Dabei können absolute Werte (optional mit der Einheit „px“ für Pixel), relative Werte (0 - 100%) sowie der spezielle Wert „N“ verwendet werden, der „next to neighbour“ bedeutet, also das Element (Container oder Pane) unmittelbar an das Nachbar-Element anordnet. |
| `dx, dy (S)` | Breite („w“) bzw. Höhe („h“) des Containers bzw. der Pane. Dabei können absolute Werte (optional mit der Einheit „px“ für Pixel), relative Werte (0 - 100%) sowie der spezielle Wert „*“ verwendet werden, der „rest of the available area“ bedeutet, also das Element (Container oder Pane) bis zum Rand des umgebenden Elements streckt. |
| `split (E)` | (Optional) Definiert, ob der Container über eine horizontale (split=“H“) oder vertikale (split=“V“) Splitbar verfügen soll. Standardwert ist keine Splitbar. |
| `fixed (B)` | (Optional) Definiert, dass eine Splitbar vorhanden, jedoch funktionslos ist; optisch werden also die nebeneinander bzw. übereinander stehenden Panes mit einem definierten Abstand voneinander dargestellt. |

### API-Funktionen

Keine.

### Beispieldarstellung

```
<bw:portal id="mainportal">
  <bw:portletcontainer id="c1" px="0" py="0" dx="100%" dy="100%" split="V">
    <bw:portletpane id="navi" px="0" py="0" dx="200px" dy="100%" radio="1"/>
    <bw:portletpane id="cont" px="N" py="0" dx="*" dy="100%" radio="0"/>
  </bw:portletcontainer>
</bw:portal>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.17.2 <bw:portletpane>

### Kurzbeschreibung

Eine `<bw:portletpane>` ist ein spezielles Layout-Element zur Anordnung von Portlets innerhalb eines Portals.

Ein `<bw:portletpane>`-Element kann (über seine `id`) als Ziel (`target`) eines Portlets dienen. Ein Portlet, dessen `target`-Attribut mit der `id` eines `<bw:portletpane>`-Elements übereinstimmt, wird innerhalb des entsprechenden Pane-Elements dargestellt. Die Darstellung folgt dabei bestimmten Regeln, die primär vom Portletcontainer vorgegeben werden; sie kann jedoch in gewissen Grenzen durch das Portlet selbst bestimmt werden (z.B. kann das Portlet eine Höhe vorgeben, die jedoch ggf. vom Portletcontainer ignoriert werden kann !).

Innerhalb einer Portletpane können theoretisch beliebig viele Portlets eingefügt werden; falls keine besonderen Anweisungen mittels Attributen erfolgen, teilen sich die Portlets die Pane gleichmäßig (bezüglich Höhe bzw. Breite).

### Syntax

```
<bw:portal
  id="str_widget_id"
  min="int_minsize"
  >
  <bw:portletcontainer
    id="str_widget_id"
    px="pos-x"
    py="pos-y"
    dx="width"
    dy="height"
    split="split-orientation"
    />
  <bw:portletpane
    id="widget-id"
    px="pos-x"
    py="pos-y"
    dx="width"
    dy="height"
    radio="radio-mode"
    />
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
</bw:portal>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine div-Section und muss daher auch eindeutig in Bezug auf alle anderen Ids der BML-Seite sein. Die ID einer Pane ist dabei die Zieladresse eines Portlets (siehe auch `target`-Attribut eines Portlets). |
| `px, py (S)` | Start-Koordinaten (X- bzw. Y-Position) des Containers bzw. der Pane. Dabei können absolute Werte (optional mit der Einheit „px“ für Pixel), relative Werte (0 - 100%) sowie der spezielle Wert „N“ verwendet werden, der „next to neighbour“ bedeutet, also das Element (Container oder Pane) unmittelbar an das Nachbar-Element anordnet. |
| `dx, dy (S)` | Breite („w“) bzw. Höhe („h“) des Containers bzw. der Pane. Dabei können absolute Werte (optional mit der Einheit „px“ für Pixel), relative Werte (0 - 100%) sowie der spezielle Wert „*“ verwendet werden, der „rest of the available area“ bedeutet, also das Element (Container oder Pane) bis zum Rand des umgebenden Elements streckt. |
| `radio (E)` | (Optional) Definiert (wenn true bzw. 1) einen besonderen Modus einer Pane, in dem lediglich eines der ggf. mehreren innerhalb der Pane enthaltenen Portlets maximiert sein kann; alle anderen Portlets werden dabei minimiert. Standardwert ist false bzw. 0. Dieses Verhalten wird bei Webcontrols manchmal auch „Accordeon“ genannt. |
| `min (I)` | (Optional) Definiert die Mindestbreite bzw. –höhe der Pane in Pixel; wird die Pane innerhalb eines Containers angezeigt, der über eine Splitbar verfügt (der Anwender kann also die Breite/Höhe der Pane ändern), kann dies nicht unter den Wert von „min“ erfolgen. |

### API-Funktionen

Keine.

### Beispieldarstellung

```
<bw:portal id="mainportal">
  <bw:portletcontainer id="c1" px="0" py="0" dx="100%" dy="100%" split="V">
    <bw:portletpane id="navi" px="0" py="0" dx="200px" dy="100%" radio="1"/>
    <bw:portletpane id="cont" px="N" py="0" dx="*" dy="100%" radio="0"/>
  </bw:portletcontainer>
</bw:portal>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.18 <bw:portlet>

## Kurzbeschreibung

Ein `<bw:portlet>` ist ein fensterähnlicher Bereich, der vom Portletcontainer verwaltet und innerhalb eines Portals (genauer: einer Portal-Pane) angezeigt wird. Portlets stellen die Basis des gesamten *B-Wise* GUI dar. Alle Dialoge oder Anzeigeseiten werden innerhalb von Portlets abgebildet. Auch PopUp-Dialoge enthalten Portlets (häufig allerdings nur eines).

Portlets werden als fensterähnliche Bereiche mit einer (optionalen) Titelzeile sowie entsprechenden Bedienelementen (Buttons zur Minimierung/Maximierung bzw. einer Border) dargestellt. Das Layout kann in gewissen Grenzen mit Hilfe von Konfigurationsdateien im Client beeinflusst werden (siehe hierzu auch Anhang G). Portlets können vom Anwender maximiert oder minimiert werden.

Beispiele für maximierte Portlets innerhalb der Abbildung 32 sind „Partnerliste“ oder das „Partnerportal“. Minimierte Portlets sind z.B. „Aktionen am Partner“ oder „Kundenspiegel“.

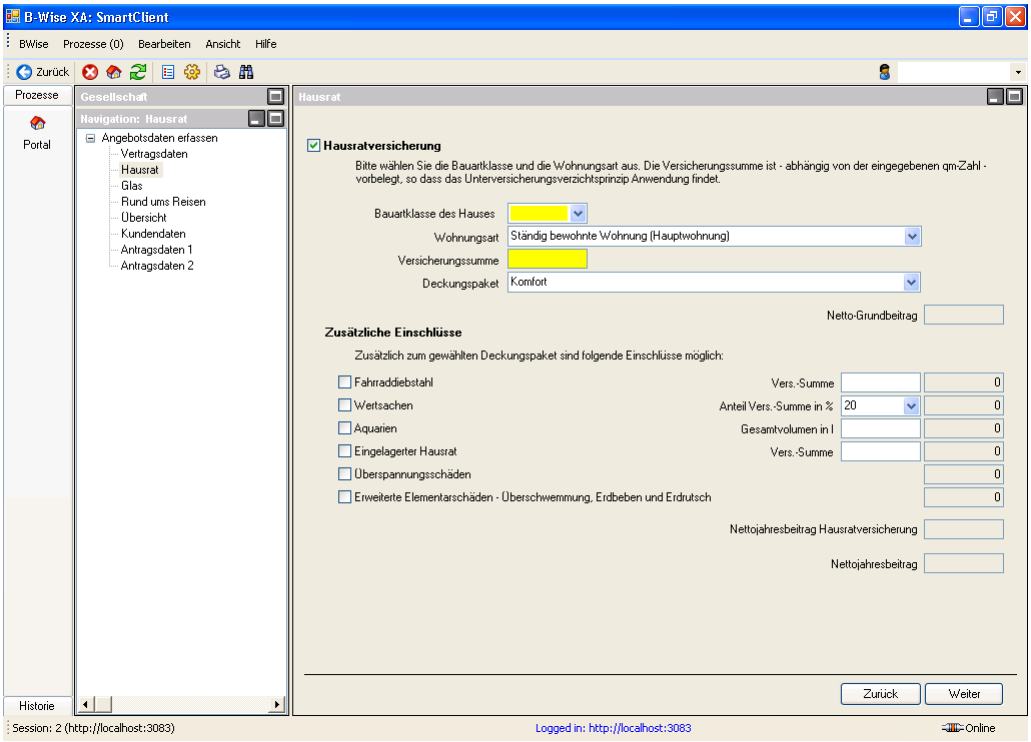



**Abb. 33: Portal mit Portlets (Beispiel SmartClient)**

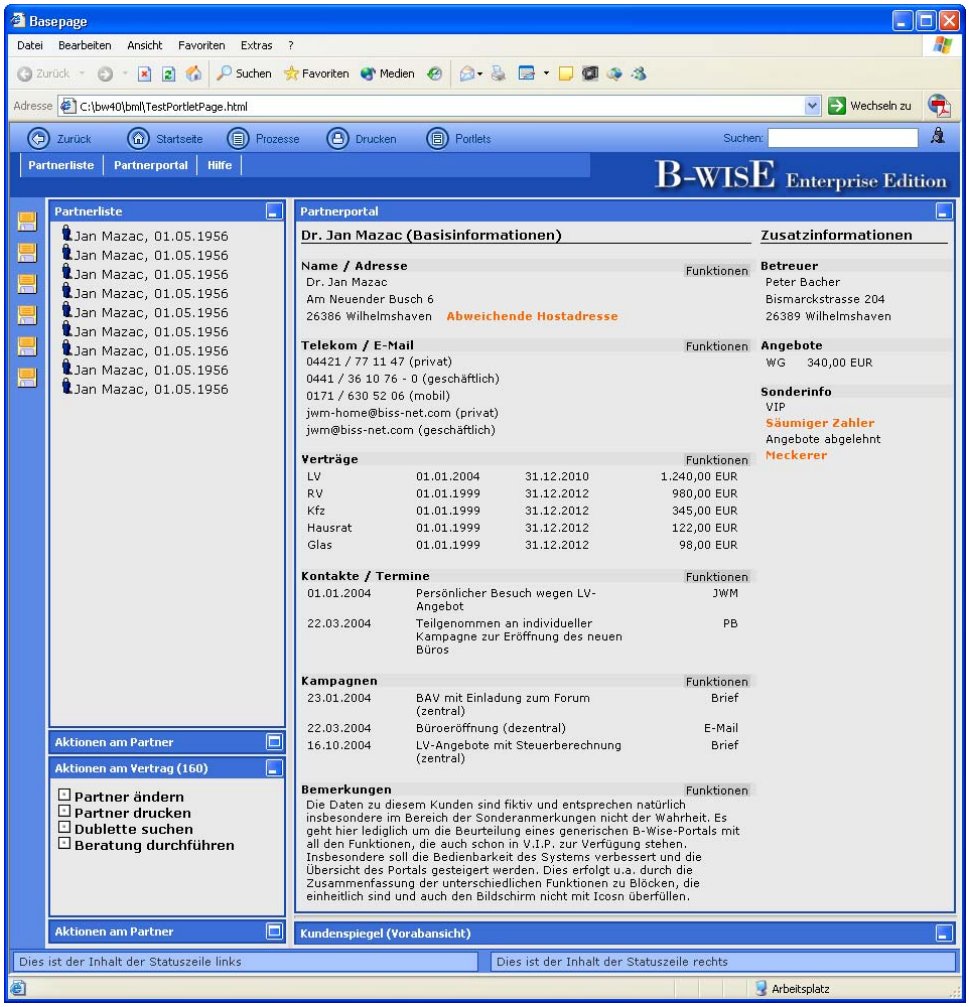
Gemäß *B-Wise* Konventionen wird jedes Portlet innerhalb einer separaten Datei definiert, die mit der Dateiendung „.plt“ endet. Selbstverständlich muss nicht die gesamte Definition in-


© Copyright BISS GmbH, 2001-2012 All Rights reserved

nerhalb einer einzige Datei enthalten sein; so können häufig Include- oder Rules-Dateien die Redundanz vermindern und damit die Wartbarkeit deutlich erhöhen.

Es ist allerdings nicht möglich, innerhalb einer Datei mehr als ein Portlet zu definieren, da es gemäß den aktuellen Konventionen eine eineindeutige Beziehung zwischen dem Dateinamen (ohne Dateierweiterung) und der Portlet-ID gibt !

Zur Laufzeit definiert der Prozess das Portallayout, in dem er die gewünschte Layout-Datei wählt. Zusätzlich definiert er die benötigten (oder gewünschten) Portlets; dies kann u.U. auch in Abhängigkeit von bestimmten Parametern erfolgen (z.B. der Gruppe des angemeldeten Benutzers). Hat der Prozess mehrere logische Schritte (Steps), so kann er diesen jeweils ein oder mehrere Portlets zuordnen. Wählt der Anwender im Anschluss einen Schritt, werden die diesem Schritt zugeordneten Portlets transformiert und innerhalb des Clients angezeigt.


Basepage
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse C:\bw40\bml\TestPortletPage.html
Wechseln zu
Zurück Startseite Prozesse Drucken Portlets Suchen:
Partnerliste Partnerportal Hilfe
B-WISE Enterprise Edition
Partnerliste
Jan Mazac, 01.05.1956
Partnerportal
Dr. Jan Mazac (Basisinformationen)
Zusatzinformationen
Name / Adresse
Funktionen
Dr. Jan Mazac
Am Neuender Busch 6
26386 Wilhelmshaven
Abweichende Hostadresse
Telekom / E-Mail
04421 / 77 11 47 (privat)
0441 / 36 10 76 - 0 (geschäftlich)
0171 / 630 52 06 (mobil)
jwm-home@biss-net.com (privat)
jwm@biss-net.com (geschäftlich)
Verträge
LV 01.01.2004 31.12.2010 1.240,00 EUR
RV 01.01.1999 31.12.2012 980,00 EUR
Kfz 01.01.1999 31.12.2012 345,00 EUR
Hausrat 01.01.1999 31.12.2012 122,00 EUR
Glas 01.01.1999 31.12.2012 98,00 EUR
Kontakte / Termine
01.01.2004 Persönlicher Besuch wegen LV-Angebot JWM
22.03.2004 Teilgenommen an individueller Kampagne zur Eröffnung des neuen Büros PB
Kampagnen
23.01.2004 BAV mit Einladung zum Forum (zentral) Brief
22.03.2004 Büroeröffnung (dezentral) E-Mail
16.10.2004 LV-Angebote mit Steuerberechnung (zentral) Brief
Bemerkungen
Die Daten zu diesem Kunden sind fiktiv und entsprechen natürlich insbesondere im Bereich der Sonderanmerkungen nicht der Wahrheit. Es geht hier lediglich um die Beurteilung eines generischen B-Wise-Portals mit all den Funktionen, die auch schon in V.I.P. zur Verfügung stehen. Insbesondere soll die Bedienbarkeit des Systems verbessert und die Übersicht des Portals gesteigert werden. Dies erfolgt u.a. durch die Zusammenfassung der unterschiedlichen Funktionen zu Blöcken, die einheitlich sind und auch den Bildschirm nicht mit Icons überfüllen.
Betreuer
Peter Bacher
Bismarckstrasse 204
26389 Wilhelmshaven
Angebote
WG 340,00 EUR
Sonderinfo
VIP
Säumiger Zahler
Angebote abgelehnt
Meckerer
Aktionen am Partner
Aktionen am Vertrag (160)
Partner ändern
Partner drucken
Dublette suchen
Beratung durchführen
Aktionen am Partner
Kundenspiegel (Vorabansicht)
Dies ist der Inhalt der Statuszeile links
Dies ist der Inhalt der Statuszeile rechts
Arbeitsplatz

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Abb. 34: Portal mit Portlets (Beispiel BrowserClient)**

### Syntax

```
<bw:portlet
   id="str_widget_id"
   target="str_pane_name"
   title="str_title_text"
   vstate="enum_state"
   mode="enum_portlet_mode"
   titlebar="bool"
   position="int_relpos"
   width="int_w"
   height="int_h"
   >

   ... Portlet-Inhalt (Controls bzw. BML-Code) ...

</bw:portlet>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `ifocus (S)` | (Optional) Definiert die ID desjenigen Elements innerhalb des Portlets, welches initial den Fokus bekommen soll. |
| `title (S)` | (Optional) Definietr den Titel des Portlets, der in der Tutlebar des Portlets angezeigt wird; um den Titel anzuzeigen muss das Attribut „titlebar“ den Wert „true“ haben. |
| `target (S)` | Definiert die ID derjenigen Pane, in die das Portlet eingefügt werden soll. |
| `vstate (E)` | (Optional) Definiert den initialen Zustand des Portlets nach Einfügen in die Pane; mögliche Wrete sind: „normal“, „min“ oder „max“. |
| `position (I)` | (Optional) Definiert die bevorzugte Position des Portlets innerhalb der Pane; je höher der Wert, umso tiefer (oder weiter rechts) wird das Portlet positioniert; dies gilt natürlich nur für den fall, dass sich mehrere Portlets innerhalb einer Pane befinden. |
| `mode (E)` | (Optional) Definiert das Verhalten des Portlets; mögliche Werte sind: „normal“ oder „sticky“; bei Angabe von „sticky“ bleibt das Portlat auch beim |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | „removePortlets“-Command vom Server erhalten; Default ist „normal“. |
| `width (I)` | (Optional) Definiert die Breite des Portlets in Pixel. In diesem Fall wird das Portlet exakt in der angegebenen Breite angezeigt, unabhängig von der Größe der Pane. |
| `height (I)` | (Optional) Definiert die Höhe des Portlets in Pixel. In diesem Fall wird das Portlet exakt in der angegebenen Höhe angezeigt, unabhängig von der Größe der Pane. |
| `titlebar (B)` | (Optional) Falls „false“, enthält das Portlet keine Titlebar; Default ist „true“. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`minimize()`**

Minimiert das Portlet.

**`maximize()`**

Maximiert das Portlet.

**`close()`**

Löscht/Schließt das Portlet. Diese Funktion entspricht der Funktion `removePortal( id)` des Portletcontainers.

**`addControls( markup )`**

Fügt die als `markup` übergebenen Controls zum Portlet hinzu. Bei `markup` muss es sich um die BML-Definition der gewünschten Controls handeln, inkl. aller benötigten Attribute und Child-Elemente.

**`removeControls( idlist )`**

Löscht alle Controls innerhalb des Portlets, deren ID innerhalb der als Argument übergebenen `idlist` vorhanden ist.

**`setFocus()`**

Setzt den Focus auf das erste Widget (dasjenige mit der niedrigsten Tab-ID) innerhalb des Portlets.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setTitle( txt )`**

Setzt den Titel des Portlets auf den als `txt` übergebenen Wert (der Titel eines Portlets ist der Text innerhalb der Portlet-Titlebar).

### Beispieldarstellung

```
<bw:portlet
   id="Test_Browser"
   title="Beispiel-Portlet"
   position="22"
   target="cont">

  <bw:browser id="BR_1" tabindex="1">
   <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
            eox="0" eoy="0" ecx="0" ecy="0"/>
   <bw:content>
     <bw:include src="Test_SampleShow.bml" />
   </bw:content>
  </bw:browser>

</bw:portlet>
```

In der Abbildung 32 und 33 sind typische Portlets abgebildet.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.19 <bw:barmenu>, <bw:contextmenu>, <bw:menu>

### Kurzbeschreibung

Ein `<bw:barmenu>` ist ein sog. PullDown-Menü, also ein Menü, das als Leiste mit waagerecht nebeneinander stehenden Optionen angezeigt wird, wobei evtl. vorhandene Unteroptionen (Sub-Item) als „PullDown"-Fenster ausgeführt sind. Ein BarMenu kann innerhalb von B-Wise überall vorkommen und eine beliebige Größe haben (es muss also nicht zwingend nur am oberen Rand eines Portlets platziert sein und sich über die gesamte Breite des Portlets erstrecken. Auf Grund der Bedien-Konventionen der Basisplattform ist es jedoch ratsam, die Positionen eines Barmenüs sehr sorgfältig zu wählen.

Ein `<bw:contextmenu>` ist ein PopUp-Menü, also ein Menü, dass beim Druck auf die rechte Maustaste erscheint. Ein `<bw:contextmenu>` kann jedem Control zugeordnet werden (also z.B. einem Browser-Element oder einem List-Element).

**Hinweis:** Innerhalb der Projekt-Konventionen sollte festgelegt werden, wie mit diesen Elementen umgegangen werden soll. So kann z.B. in einem Projekt festgelegt werden, dass auf Kontextmenüs grundsätzlich verzichtet wird (also keine Nutzung der rechten Maustaste) und dass ein Barmenü ausschließlich als client-weites Menü verwendet wird. Damit wären diese Elemente in einem derartigen Projekt nicht verwendbar.

Sowohl ein `<bw:barmenu>` als auch ein `<bw:contextmenu>` sind Container für das eigentliche Menü-Element (<bw:menu>), welches die Menü-Optionen, die zugehörigen Commands sowie ggf. Icons bzw. Accel-Keys definiert.

ID-Konvention: „BM_" für BarMenu und „CM_" für ContextMenu (popUp-Menu).

### Syntax

```
<bw:barmenu
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      rendermode="enum_mode"
      >
      <bw:menu
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
            id="str_widget_id"
            styleref="str_tstyle_reference"
            action="str_command"
            >
            <bw:mi
                  id="str_widget_id"
                  accel="acc_text"
                  imgSrc="images_src"
                  imgRes="image_id"
                  >
                  Menutext
                  <bw:events>
                        <bw:event ...>
                  </bw:events>
            </bw:mi>
            <bw:msep />
            <bw:msub
                  id="widget-id"
                  label="menu-text"
                  >
                  <bw:mi
                        id="str_widget_id"
                        accel="acc_text"
                        imgSrc="images_src"
                        imgRes="image_id"
                        >
                        Menutext
                        <bw:events>
                              <bw:event ...>
                        </bw:events>
                  </bw:mi>
            </bw:msub>
      </bw:menu>
</bw:barmenu>
```

**Hinweis:** Anstelle von `<bw:barmenu>` kann in identischer Form ein `<bw:contextmenu>` verwendet werden.

### Parameter

| Paremeter | Beschreibung |
|---|---|


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine div-Section und muss daher auch eindeutig in Bezug auf alle anderen Ids der BML-Seite sein. Alle Menü-Items haben eine ID, über die sie auch innerhalb des APIs (siehe unten) angesprochen werden können. |
| `style (S)` | Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.6 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.6 „Styles“ aufgeführt. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist |
| `enabled (B)` | (Optional) Erlaubt die Darstellung als aktives (`enabled="true"`) oder inaktives (`enabled="false"`) Element. Standard-Einstellung ist `enable="true"`. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `label(S)` | Definiert den Text, der innerhalb des GUI für dieses Menü-Item erscheinen soll. |
| `accel (S)` | (Optional) Definiert einen Text, der rechtsbündig innerhalb der Menüoption angezeigt wird und normalerweise den sog. „Accelerator-Key“ kennzeichnet, also diejenige Tasten-Kombination, die alternativ zur Menüoption dieselbe Aktion auf dem B-Wise Server auslöst (z.B. „Strg-X“ für den Accel-Key). Erlaubte Werte sihe Abschnitt 9.X. |
| `imgSrc (S)` | (Optional) Zeigt links vom Text (am linken Rand) das angegebene Image an. Das exakte Layout kann anwendungsweit innerhalb des jeweiligen Clients bestimmt werden (z.B. grauer Hintergrund der Spalte mit den Images) |
| `imgRes (S)` | (Optional) Definiert eine ID (String), die ein Icon innerhalb der Ressource-Datei des SmartClients kennzeichnet; dieses Attribut wird nur vom SmartClient ausgewertet; die ID muss auf ein Icon zeigen, das in der entsprechenden Ressource-Datei auch vorhanden ist ! |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des `<bw:style>`-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs. `<bw:style>`-Tags sind im `<bw:barmenu>`-Element erlaubt. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). Das `<bw:pos>`-Tag ist nur als Child des `<bw:barmenu>`- bzw. des `<bw:contextmenu>`-Elements erlaubt. Wird es für ein Menü vom Typ |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | `<bw:contextmenu>` definiert, wird es allerdings ignoriert, da dieses immer an der aktuellen Mausposition angezeigt wird. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. Das `<bw:events>`-Tag ist im `<bw:mi>`-Element erlaubt. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip" innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5.. Ein `<bw:tooltip>`-Tag ist für `<bw:mi>`-Tags des Menüs erlaubt. |
| `<bw:msep>` | (Optional) Definiert einen Menü-Separator (eine grafische Linie, welche zwei Optionen voneinander trennt). |
| `<bw:msub>` | (Optional) Definiert einen Submenu-Block, der innerhalb als eigene Menü-Pane mit den darin enthaltenen Optionen dargestellt wird (siehe auch Beispiel-Abbildung weiter unten). Ein `<bw:msub>` sollte zwingend ein `label`-Attribut haben, das den Text des Untermenü-EIntrags darstellt. |
| `<bw:mi>` | (Optional) Definiert ein Menu-Item, also einen Menü-Eintrag. Ein `<bw:mi>` hat folgende Attribute:<br>`id` = Eindeutige ID des Menü-Eintrags (String)<br>`label` = Text des Menü-Eintrags<br>`accel` = (Optional) Einen zugeordneten Accelerator-Key (z.B. „Strg-K")<br>`imgSrc` = (Optional) Ein zugeordnetes Image/Icon, das in der linken Spalte des Menüs angezeigt wird<br>`imgRes` = (Optional) Nur für den SmartClient; die ID (String) eines Icons, welches aus der Ressource-Datei des SmartClients geladen werden kann |
| `<bw:mci>` | |

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setEnabled( state )`**

Setzt den Zustand des Elements auf den logischen Wert „enabled" (also aktiv, falls `state = true`) oder „disabled" (also inkativ, falls `state = false`).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate []`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element abhängig. Dieses Event wird beim Label nicht ausgewertet.

**`mouseenter []`**

Getriggert, wenn der Mauszeiger das „bounding rectange“ eines Menü-Elements (menu items) betritt.

**`mouseexit []`**

Getriggert, wenn der Mauszeiger das „bounding rectangle“ eines MenüElements (menu items) verlässt.

### Beispieldarstellung

```
<bw:compound id="CP_1" tabindex="1" border="Flat">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="50" eoy="50" ecx="500" ecy="150"/>
  <bw:barmenu id="BM_2">
    <bw:pos aox="0" aoy="0" acx="1000" acy="0"
           eox="0" eoy="0" ecx="0" ecy="21"/>
    <bw:menu>
      <bw:msub label="B-Wise">
        <bw:mi label="Testen" accel="C+T" imgSrc="img/AddMem.png" />
        <bw:mi label="Ausliefern" accel="C+A" imgSrc=" img/DelMem.png" />
        <bw:mci label="Pflegen" accel="C+D" />
        <bw:msep />
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
        <bw:mi label="Beenden" accel="A+F4" action="B_END()" />
      </bw:msub>
    </bw:menu>
  </bw:barmenu>

  <bw:label id="LA_1a">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="10" eoy="45" ecx="90" ecy="66"/>
      <bw:content>
        <bw:t>Dialogblock 1</bw:t>
      </bw:content>
  </bw:label>
  <bw:entry id="TF_1a" tabindex="2">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="100" eoy="45" ecx="400" ecy="66"/>
    <bw:contextmenu id="CM_Entry">
      <bw:menu>
        <bw:mi label="Testen" accel="C+T" imgSrc="img/AddMem.png" />
        <bw:mi label="Ausliefern" accel="C+A" imgSrc="img/DelMem.png" />
        <bw:mci label="{Modell.LabelValue}" accel="C+D" />
        <bw:msep />
        <bw:mi label="Beenden" accel="A+F4" action="B_END()" />
      </bw:menu>
    </bw:contextmenu>
  </bw:entry>
</bw:compound>
```

Selbstverständlich kann als Inhalt des `label`-Attributs auch ein variabler Text bzw. der Inhalt eines Objektmodell-Elements verwendet werden verwendet werden. Dies ist im obigen Beispiel für das dritte Element des Kontext-Menüs dargestellt.

Die folgende Abbildung enthält zwei typische Bar-Menüs (einmal für das Portlet, einmal für einen `<bw:compound>`-Bereich) sowie ein Kontext-Menü, das einem Eingabefeld zugeordnet worden ist.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

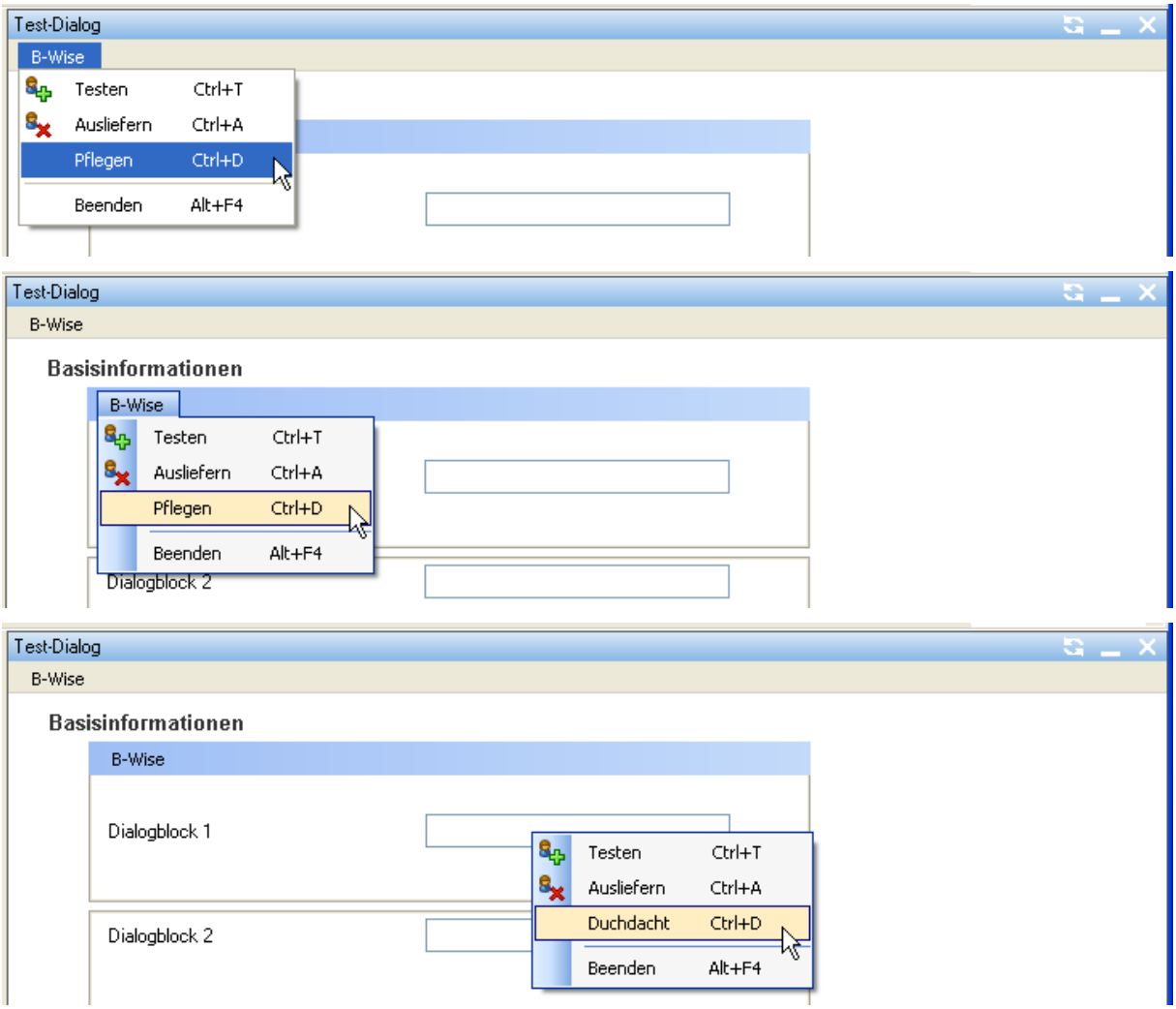



**Abb. 35: Beispiele für Menü-Elemente**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.20 <bw:compound>

**Kurzbeschreibung**

Ein <bw:compound> ist ein Layoutelement, welches eine logische Gruppe sowie gleichzeitig einen physischen (Bildschirm-) Bereich darstellt. Es dient primär der Zusammenfassung anderer Elemente zu logischen und visuellen Einheiten. Logisch in dem Sinne, dass z.B. ein „disable“ auf das Compound-Element alle darin enthalenen Elemente disabled, visuell in dem Sinne, dass sich z.B. die Layout-Positionen der Elemente auf das Compound-Element beziehen.

Eine der Eigenschaften eines <bw:compound>-Elements ist die Scroll-Eigenschaft; ist der innerhalb des Elements definierte Inhalt in seiner Bildschirmausdehnung größer als der innerhalb des <bw:compound>-Elements zur Verfügun stehende Platz (definiert durch die Abmaße des <bw:compound>-Elements), werden vertikale und/oder horizontale Scrollbars dargestellt und der Anwender kann zu jedem beliebigen Teil des Inhalts scrollen.

Ein <bw:compound>-Element ist ebenfalls bei der relative Positionierung von Gruppen innerhab eines Dialogs hilfreich. Zusätzlich ist es in vielen Situationen nützlich, Control-Elemente zu Gruppen zusammenzufassen, die gemeinsam sichtbar bzw. unsichtbar gesetzt werden sollen. Auch dies kann mit einem <bw:compound>-Element einfach bewerkstelligt werden.

Schließlich kann ein <bw:compound>-Element bei Teiltransformationen hilfreich sein. Dabei definiert es eine Gruppe visueller Elemente, die für sich auf einem B-Wise Server transformiert und zum Client übertragen werden können. Teiltransformationen („DOM-Fragmente“) sparen Bandbreite und erlauben ein weitestgehend „flickerfreies“ Arbeiten.

Abb. 35 veranschaulicht das entsprechende, visuelle Layout.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

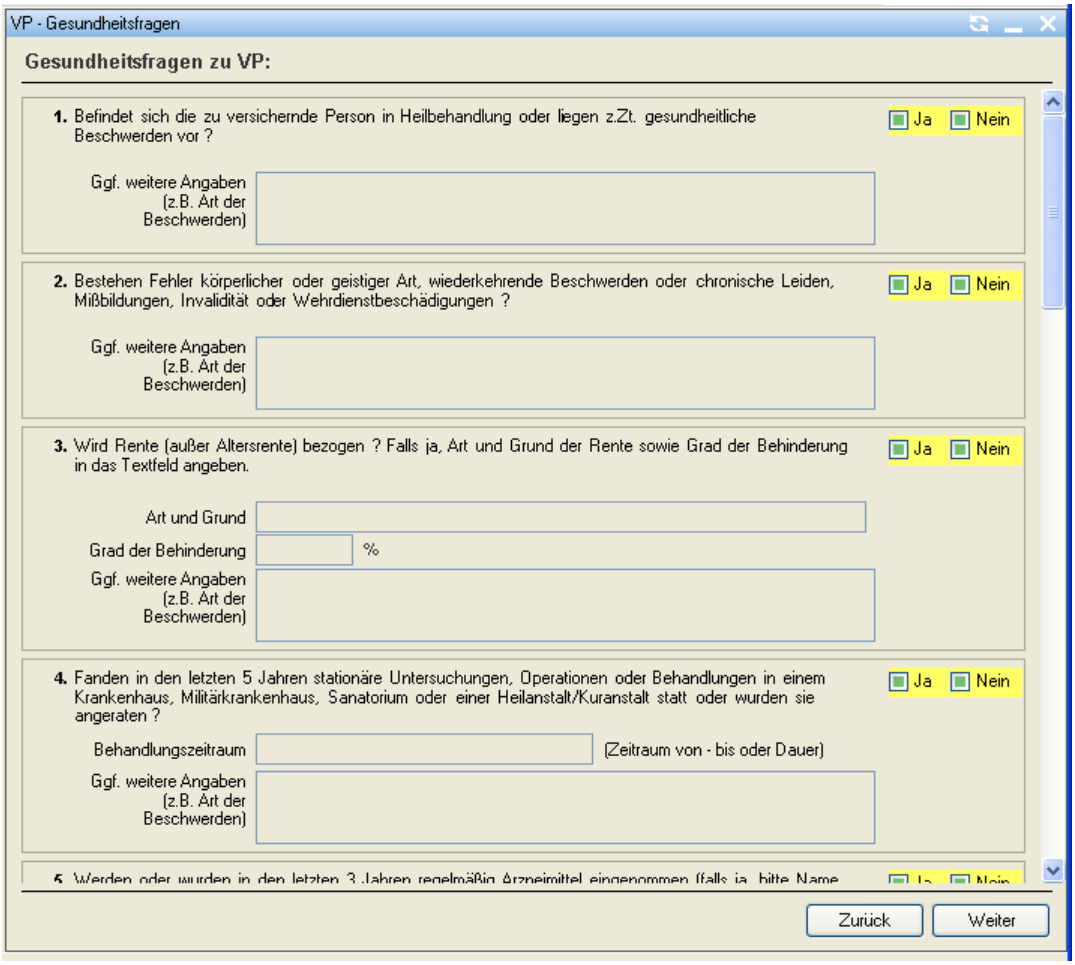

VP - Gesundheitsfragen
Gesundheitsfragen zu VP:
1. Befindet sich die zu versichernde Person in Heilbehandlung oder liegen z.Zt. gesundheitliche Beschwerden vor ?
Ja
Nein
Ggf. weitere Angaben (z.B. Art der Beschwerden)
2. Bestehen Fehler körperlicher oder geistiger Art, wiederkehrende Beschwerden oder chronische Leiden, Mißbildungen, Invalidität oder Wehrdienstbeschädigungen ?
Ja
Nein
Ggf. weitere Angaben (z.B. Art der Beschwerden)
3. Wird Rente (außer Altersrente) bezogen ? Falls ja, Art und Grund der Rente sowie Grad der Behinderung in das Textfeld angeben.
Ja
Nein
Art und Grund
Grad der Behinderung
%
Ggf. weitere Angaben (z.B. Art der Beschwerden)
4. Fanden in den letzten 5 Jahren stationäre Untersuchungen, Operationen oder Behandlungen in einem Krankenhaus, Militärkrankenhaus, Sanatorium oder einer Heilanstalt/Kuranstalt statt oder wurden sie angeraten ?
Ja
Nein
Behandlungszeitraum
(Zeitraum von - bis oder Dauer)
Ggf. weitere Angaben (z.B. Art der Beschwerden)
Zurück
Weiter

**Abb. 36: Beispiel von Compound-Elementen mit vertikaler Scrollbar**

Die Abbildung zeigt einen Dialog, der insgesamt deutlich „länger“ ist, als der zur Verfügung stehende Platz. Die einzelnen Fragen-Elemente (ihrerseits ebenfalls `<bw:compounds>`), die sich innerhalb des Dialogs befindet, können mit Hilfe der Scrollbar erreicht werden. Darüberhinaus ist es in diesem Dialog möglich, durch markieren des „Ja“-Kästchens weitere Inhalte darzustellen; auch diese befinden sich innerhalb eines weiteren Compound-Elements und werden so „ein- bzw. ausgeschaltet“.

Bei `<bw:compound>`-Elementen macht auch die relative Positionierung Sinn (siehe auch Abschnitt 9.8.2); auch diese Funktionalität ist ebenfalls in der Abbildung 35 (allerdings nicht sichtbar) dargestellt. Die einzelnen Fragen-Elemente werden bei diesem Dialog dynamisch anhand einer im Prozess definierten (und vom jeweils gewählten Tarif abhängigen) Liste dargestellt. Somit ist beim Dialog-Design nicht klar, an welche absolute Position diese Elemente platziert werden können. Jede Frage (d.h. alle zur Frage gehörenden Controls) sind


© Copyright BISS GmbH, 2001-2012 All Rights reserved

innerhalb eines `<bw:compound>`-Elements enthalten; das jeweilige `<bw:compound>`-Element wiederum ist relativ zum vorherigen mit einem Abstand von 5 Pixel positioniert.

ID-Konvention: „CP_“

### Syntax

```
<bw:compound
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      border="enum_style"
      >

      ... Alle BML-Controls ...

</bw:compound>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurück- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | setzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible="`true`") oder nicht (visible="`false`"); Standard-Einstellung ist visible="`true`". |
| `rolocked (B)` | (Optional) Wenn „true", verhindert Hittesting und die Erkennung von Events im „readonly"-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly"-Zustand auf „mouse-over"-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true", erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock"-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset") um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `scroll (E)` | (Optional) Definiert, wie sich das Compound verhält, falls der Inhalt größer ist als die Maße des Compounds; mögliche Werte sind: „none" (Inhalt wird abgeschnitten), „horizontal" (ggf. wird eine horizontale Scrollbar dargestellt, vertikaler Inhalt wird ggf. abgeschnitten), „vertical" (ggf. wird eine vertikale Scrollbar dargestellt, horizontaler Inhalt wird ggf. abgeschnitten), „both" (ggf. werden beide Scrollbars dargestellt, sodass jeglicher Inhalt ggf. durch Scrollen erreicubar ist). |
| `ifocus (S)` | (Optional) Definiert die ID desjenigen Elements, welches initial den Fokus erhalten soll. **Hinweis:** Auch wenn dieses Attribut auch beim Compound definiert ist, macht es nur beim Portlet Sinn und sollte daher im Zusammenhang mit einem Compound nicht verwendet werden! |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsen- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

tiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden.

`<bw:...>` Siehe Kurzdoku.

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`addControls( ctrls)`**

Fügt die Elemente „ctrls“ in das Compound hinzu.

**`removeControls( id)`**

Löscht die Controls unterhalb und inkl. des Elements mit der ID „id“.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**Layouted []**

Getriggert, nachdem das Compound und seine Inhalte auf dem Bildschirm positioniert worden sind.

### Beispieldarstellung

```
<bw:compound id="CP_Main">
  <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
           eox="0" eoy="35" ecx="0" ecy="-45"/>

  <bw:var name="qnum" expr="0"/>
  <bw:foreach
    element="ObjectWrapper.VPListe.*VP[pos() eq {AP}].GFListe.*GF">
    <bw:if expr="( $versArt eq 'Vollversicherung' and .Voll = true">
      <bw:var name="qnum" expr="$qnum+1"/>
      <bw:var name="cqc" expr="current()"/>
      <bw:include src="{concat( 'Antrag_VP', .QID, '.inc')}" />
    </bw:if>
  </bw:foreach>

</bw:compound>
```

Das dargestellte `<bw:compound>`-Element ist der “Container” für die in Abbildung 35 gezeigten Gesundheitsfragen eines typischen Antragsformulars. In diesem kurzen BML-

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Code wird ebenfalls deutlich, wie effizient auch komplexe, dynamische Sachverhalte mit Hilfe von BML formuliert werden können.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.21 <bw:compoundbox>

### Kurzbeschreibung

Eine `<bw:compoundbox>` entspricht einem `<bw:compound>`-Element bei dem zusätzlich ein Rechteck mit einem optionalen Text dargestellt wird. Alle anderen Eigenschaften teilt die `<bw:compoundbox>` mit dem `<bw:compound>`-Element (von dem sie auch technisch abgeleitet ist).

Eine `<bw:compoundbox>` wird insbesondere zusammen mit Radiobuttons verwendet (siehe auch Abschnitt 9.5.X), für die sie einen Container darstellt (Radiobuttons benötigen eine logische Gruppierung und die das `<bw:compoundbox>`-Element bildet hierfür die Basis).

### Syntax

```
<bw:compoundbox
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      title="str_text"
      >

      ... Alle BML-Controls ...

</bw:compoundbox>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `scroll (E)` | (Optional) Definiert, wie sich das Compound verhält, falls der Inhalt größer ist als die Maße des Compounds; mögliche Werte sind: „none“ (Inhalt wird abgeschnit- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | ten), „horizontal“ (ggf. wird eine horizontale Scrollbar dargestellt, vertikaler Inhalt wird ggf. abgeschnitten), „vertical“ (ggf. wird eine vertikale Scrollbar dargestellt, horizontaler Inhalt wird ggf. abgeschnitten), „both“ (ggf. werden beide Scrollbars dargestellt, sodass jeglicher Inhalt ggf. durch Scrollen erreicubar ist). |
| `ifocus (S)` | (Optional) Definiert die ID desjenigen Elements, welches initial den Fokus erhalten soll. **Hinweis:** Auch wenn dieses Attribut auch beim Compound definiert ist, macht es nur beim Portlet Sinn und sollte daher im Zusammenhang mit einem Compound nicht verwendet werden! |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:...>` | Siehe Kurzdoku. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`addControls( ctrls)`**

Fügt die Elemente „ctrls“ in das Compound hinzu.

**`removeControls( id)`**

Löscht die Controls unterhalb und inkl. des Elements mit der ID „id“.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**Layouted []**

Getriggert, nachdem das Compound und seine Inhalte auf dem Bildschirm positioniert worden sind.

## Beispieldarstellung

```
<bw:compoundbox id="CP_RBTest" tabindex="1" title="Geschlecht">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="30" eoy="30" ecx="670" ecy="200"/>
  <bw:radiobutton id="RB_1" datasrc="">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="20" ecx="200" ecy="41"/>
    <bw:content>
      <bw:t>männlich</bw:t>
    </bw:content>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  </bw:radiobutton>
  <bw:radiobutton id="RB_2" datasrc="">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="45" ecx="200" ecy="66"/>
    <bw:content>
      <bw:t>weiblich</bw:t>
    </bw:content>
  </bw:radiobutton>
</bw:compound>
```

Die abgebildete Compoundbox dient als Container für die beiden Radiobuttons.

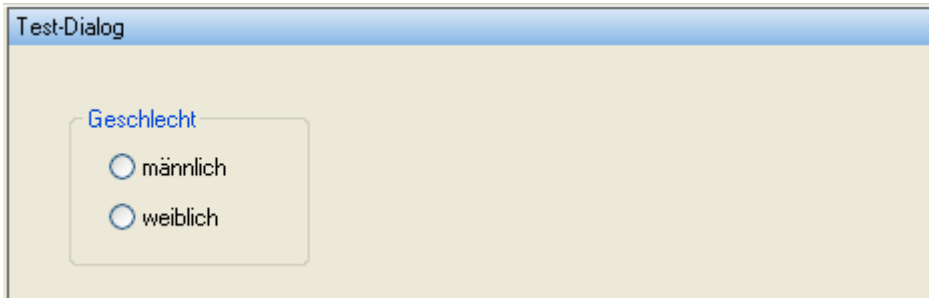



**Abb. 37: Beispiel für eine <bw:compoundbox>**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22 <bw:toolbar>

### Kurzbeschreibung

Eine <bw:toolbar> ist ein Bereich innerhalb eines Fensters oder einer Pane, der normalerweise Buttons, Choices oder Entryfields enthält, die der einfachen und schnellen Ausführung von Befehlen dienen sollen. Die folgende Abbildung zeigt eine typische Toolbar.

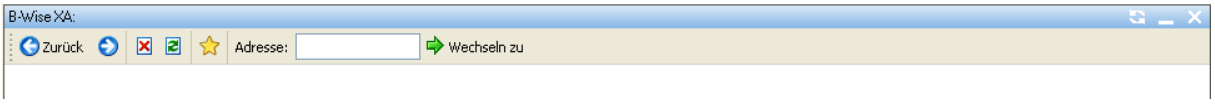



**Abb. 38: Beispiel einer Toolbar**

Die Toolbar dient somit als Container für andere Widgets, ist jedoch selbst ein sichtbares Element. Ein Fenster oder eine Pane können mehr als nur eine Toolbar enthalten; dies ermöglicht die logische Gruppierung zusammen gehörender (Bedien-)Element in eine Toolbar.

Eine Besonderheit der Toolbar ist, dass sie angedockt (an eine der 4 Seiten eines Portlets) oder frei schwebend (quasi als eigenes Fenster) dargestellt werden kann. Zwar kann eine Toolbar auch als Kind-Element eines `<bw:compound>`-Elements verwendet werden, die Andockbarkeit oder die Möglichkeit, die Toolbar frei schwebend darzustellen ist jedoch nur dann gegeben, wenn die Toolbar das Kind-Element eines Portlets ist.

ID-Konvention: „TB_“

### Syntax

```
<bw:toolbar
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      >
      <bw:pos ... />
      <bw:tbbutton
            id="str_widget_id"
            >
            <bw:val>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
                  Button-Text
            </bw:val>
            <bw:img src="image_src" />
      <bw:tbcbutton
            id="str_widget_id"
            checked="bool"
            >
      <bw:tbchoice
            id="str_widget_id"
            >
            <bw:items />
      <bw:tbentry
            id="str_widget_id"
            >
            <bw:val>
                  Entry-Text
            </bw:val>
      <bw:tblabel
            id="str_widget_id"
            >
            <bw:val>
                  Label-Text
            </bw:val>
      <bw:tbmenu
            id="str_widget_id"
            >
            <bw:items ... />
            <bw:events ... />
      <bw:tbmbutton
            id=""
            >
            <bw:tbmenu ... />
      <bw:tbseparator />
</bw:toolbar>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.6 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.6 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `rendermode (E)` | (Optional) Definiert das Aussehen der Toolbar; mögliche Werte sind: „professional“ und „system“. |
| `gripstyle (E)` | (Optional) Definiert, ob ein sichtbares „Handle“ dargestellt wird, mit dessen Hilfe die Toolbar vom Anwender repositioniert werden kann; mögliche Werte sind: „visisble“ oder „hidden“. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |
| `<bw:tbbutton>` | Ein Button mit Text und/oder Image zum Auslösen von Befehlen, ohne Zustand. |
| `<bw:tbcbutton>` | Ein Button mit Text und/oder Image zum Auslösen von Befehlen, mit Zustand („Checkbox“-Button). |
| `<bw:tbmbutton>` | Ein Button, der ein Pulldown-Menü anzeigt. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `<bw:tblabel>` | Ein Text. |
| `<bw:tbchoice>` | Eine ComboBox (Choice) zur Auswahl eines Wertes aus einer Liste möglicher Werte. |
| `<bw:tbentry>` | Ein Eingabefeld zur Eingabe beliebiger Werte. |
| `<bw:tbseparator>` | Eine visuelle Trennung (vertikale Linie). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`removeIds( ids)`**

Löscht die Elemente mit den IDs „ids“ aus der Toolbar.

**`addElements( elems)`**

Fügt die Elemente „elems“ rechts in die Toolbar hinzu.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

### Beispieldarstellung

```
<bw:toolbar id="TB_1" rendermode="system" enabled="false">
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
          eox="0" eoy="0" ecx="0" ecy="30" />
  <bw:tbbutton>
    <bw:img src="images/BrowserBack.png" />
    <bw:val><bw:t>Zurück</bw:t></bw:val>
    <bw:events>
      <bw:event name="click" method="{event( 'DOTHIS')}" />
    </bw:events>
  </bw:tbbutton>
  <bw:tbseparator />
  <bw:tblabel>
    <bw:val>Adresse:</bw:val>
  </bw:tblabel>
  <bw:tbentry>
    <bw:val>Beispiel-Inhalt</bw:val>
  </bw:tbentry>
  <bw:tbchoice>
    <bw:items>
      <bw:item>Erste Wahl</bw:item>
      <bw:item>Zweite Wahl</bw:item>
      <bw:item>Dritte Wahl</bw:item>
    </bw:items>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  </bw:tbchoice>
  <bw:tbcbutton>
    <bw:img src="images/BrowserFavourites.png" />
  </bw:tbcbutton>
</bw:toolbar>
```

Eine Beispiel- Toolbar ist in Abb. 37 dargestellt.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22.1 <bw:tbbutton>

### Kurzbeschreibung

Ein `<bw:tbbutton>` ist ein Toolbar-Element, in diesem Fall ein Button zum Auslösen von Befehlen, ohne Statusanzeige (d.h. nach Betätigung/Drücken kehrt der Button in seine ursprüngliche, nicht gedrückte Position zurück).

ID-Konvention: „TBB_".

### Syntax

```
<bw:tbbutton
      id="str_widget_id"
      >
      <bw:content>
            Labelinhalt
      </bw:content>
</bw:tbbutton>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles". |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true" oder „false")

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## API-Funktionen

**`setEnabled( state)`**

Versetzt das Element in den Zustand „enabled“ (state = „true“) oder „disabled“ (state = „false“).

**`setText( text)`**

Setzt den Text des Elements auf den Wert „text“.

## Events

**`click [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

## Beispieldarstellung

Siehe Toolbar.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22.2 <bw:tbcbutton>

### Kurzbeschreibung

Ein `<bw:tbcbutton>` ist ein Toolbar-Element, in diesem Fall ein Button zum Auslösen von Befehlen, mit Statusanzeige (d.h. nach Betätigung/Drücken kehrt der Button nicht in seine ursprüngliche Position zurück, sondern bleibt gedrückt).

ID-Konvention: „TBX_“.

### Syntax

```
<bw:tbcbutton
      id="str_widget_id"
      >
      <bw:content>
            Labelinhalt
      </bw:content>
</bw:tbcbutton>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `checked (B)` | (Optional) Falls „true“, wird der Button initial in der gedrückten (checked) Position dargestellt; Default ist „false. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### API-Funktionen

**`setEnabled( state)`**

Versetzt das Element in den Zustand „enabled“ (state = „true“) oder „disabled“ (state = „false“).

**`setText( text)`**

Setzt den Text des Elements auf den Wert „text“.

### Events

**`click [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`check [ id ]`**

Getriggert, wenn der Button gedrückt worden ist.

**`uncheck [ id ]`**

Getriggert, wenn der button losgelassen worden ist.

**`checkchanged [ id ]`**

Getriggert, wenn der Button seinen Zustand geändert hat.

### Beispieldarstellung

Siehe Toolbar.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22.3 <bw:tbmbutton>

### Kurzbeschreibung

Ein `<bw:tbmbutton>` ist ein Toolbar-Element, in diesem Fall ein Button zum Auslösen eines PullDown-Menüs.

ID-Konvention: „TBM_“.

### Syntax

```
<bw:tbmbutton
      id="str_widget_id"
      >
      <bw:content>
            Labelinhalt
      </bw:content>
</bw:tbmbutton>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `<bw:tbmenu>` | Enthält die Menü-Definition des Menüs, welches durch Drücken des Buttons angezeigt wird. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### API-Funktionen

**`setEnabled( state)`**

Versetzt das Element in den Zustand „enabled“ (state = „true“) oder „disabled“ (state = „false“).

**`setText( text)`**

Setzt den Text des Elements auf den Wert „text“.

### Events

**`click [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

### Beispieldarstellung

Siehe Toolbar.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22.4 <bw:tblabel>

### Kurzbeschreibung

Ein `<bw:tblabel>` ist ein Toolbar-Element, in diesem Fall ein statisches Text.

ID-Konvention: „TBL_“.

### Syntax

```
<bw:tblabel
      id="str_widget_id"
      >
      <bw:content>
            Labelinhalt
      </bw:content>
</bw:tblabel>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional) Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

### API-Funktionen

**`setEnabled( state)`**

Versetzt das Element in den Zustand „enabled“ (state = „true“) oder „disabled“ (state = „false“).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

`setText( text)`

Setzt den Text des Elements auf den Wert „text“.

### Events

Keine.

### Beispieldarstellung

Siehe Toolbar.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22.5 <bw:tbchoice>

### Kurzbeschreibung

Eine `<bw:tbchoice>` ist ein Toolbar-Element, in diesem Fall eine ComboBox (Chocie) zur Auswahl eines Werte-Elements aus einer Lste möglicher Werte-Elemente.

ID-Konvention: „TBC_“.

### Syntax

```
<bw:tbchoice
      id="str_widget_id"
      >
      <bw:content>
            Labelinhalt
      </bw:content>
</bw:tbchoice>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `<bw:items>` | Definiert die Liste der möglichen Werte-Elemente. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## API-Funktionen

**`setEnabled( state)`**

Versetzt das Element in den Zustand „enabled“ (state = „true“) oder „disabled“ (state = „false“).

**`setText( text)`**

Setzt den Text des Elements auf den Wert „text“.

## Events

**`click [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`selected [ id ]`**

Getriggert, wenn der Benutzer ein Werte-Element aus der Liste ausgewählt hat.

## Beispieldarstellung

Siehe Toolbar.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22.6 <bw:tbentry>

### Kurzbeschreibung

Ein `<bw:tbentry>` ist ein Toolbar-Element, in diesem Fall ein Eingabefeld zur Erfassung beliebiger Werte.

ID-Konvention: „TBE_“.

### Syntax

```
<bw:tbentry
      id="str_widget_id"
      >
      <bw:content>
            Labelinhalt
      </bw:content>
</bw:tbentry>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `<bw:val>` | (Optional) Definiert einen initialen Werte für das Eingabefeld. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## API-Funktionen

**`setEnabled( state)`**

Versetzt das Element in den Zustand „enabled“ (state = „true“) oder „disabled“ (state = „false“).

**`setText( text)`**

Setzt den Text des Elements auf den Wert „text“.

## Events

**`click [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste clickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`selected [ id ]`**

Getriggert, wenn der Benutzer ein Werte-Element aus der Liste ausgewählt hat.

**`focuslost [ id ]`**

Getriggert, wenn das Control den Fokus verloren hat (der Anwender z.B. durch Drücken der TAB-Taste ein anderes Element fokussiert hat)

**`validate [ id ]`**

Getriggert, wenn der Benutzer die Eingabe mit Hilfe der „Enter“-Taste bestätigt hat.

## Beispieldarstellung

Siehe Toolbar.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.22.7 <bw:tbseparator>

### Kurzbeschreibung

Ein `<bw:tbseparator>` ist ein Toolbar-Element, in diesem Fall eine Trennlinie zwischen zwei anderen Elementen.

ID-Konvention: „TBS_“.

### Syntax

```
<bw:tbseparator
      id="str_widget_id"
      >
</bw:tbseparator>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

### API-Funktionen

Keine.

### Events

Keine.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Beispieldarstellung

Siehe Toolbar.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.23 <bw:radiobutton>

### Kurzbeschreibung

Ein `<bw:radiobutton>` ist Schaltelement, welches normalerweise für die exklusive Auswahl einer Option („1 aus n Möglichen“) verwendet wird. Hierzu sollten die einzelnen Radiobuttons entsprechend gruppiert und mit sinnvollen Texten (Label) versehen sein.

Logisch gesehen stellt eine derartige Gruppe den Inhalt einer auf Objektmodell-Seite vorhandenen Liste dar, aus der immer genau 0 oder 1 Element gewählt sein kann. Die Inhalte der Liste werden auf die einzelnen Buttons der Gruppe abgebildet. Hierzu werden die Texte der Liste als Texte der Radiobuttons gesetzt. Enthält die Liste weniger Elemente als Buttons vorhanden sind, bleiben die Buttons ohne Inhalt. Enthält die Liste mehr Elemente, werden die überzähligen Elemente verworfen. Dieses Konstrukt eignet sich somit gut für konstante, a-priori bekannte Listen bzw. z.B. für die Abbildung von Ja/Nein-Entscheidungen.

Selbstverständlich können die Texte der Radiobuttons auch explizit innerhalb des Dialogs als statische Konstanten angegeben werden.

**Hinweis:** Um die `<bw:radiobutton>`-Elemente zu einer logische Gruppe zusammenzufassen, müssen diese zwingend innerhalb eines `<bw:compoundbox>` oder eines `<bw:compound>`-Elements eingeschlossen sein. Die `<bw:radiobutton>`-Elemente selbst dürfen dabei kein `tabindex`-Attribut haben.

### Syntax

```
<bw:radiobutton
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      textalign="enum_textalign"
      checkalign="enum_checkalign"
      checked="bool"
      datasrc="model-element"
      >
      <bw:content>
            Text des Radiobuttons
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      </bw:content>
</bw:radiobutton>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `checked (B)` | (Optional) Falls „true“, ist der betreffende radiobutton markiert; Default ist „false“. |
| `group (S)` | (Optional) Definiert die ID (Name) einer Gruppe; nur die Radiobuttons einer Gruppe werden wechselseitig geschaltet! |
| `textalign (E)` | (Optional) Definiert die Ausrichtung des Label-Textes im Verhältnis zum Kästchen des Radiobuttons; mögliche Werte sind eine Kombination aus „top“, „middle“, „bottom“ und „left“, „center“, „right“, z.B. „topleft“; Default ist „topleft“ |
| `checkalign (E)` | (Optional) Definiert die Position des Kästchens in Relation zum Text; mögliche Werte sind „left“, „right“. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip" innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true" oder „false")

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**setVisible()**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**setCtxMenu()**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**setTooltip( item, tooltip )**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**setStyle( attrname, attrval )**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**setStyleRef( styleref )**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**addEvents()**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**removeEvents()**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**check()**

Markiert den Radiobutton als „angehakt“.

**setGroup( grp)**

Setzt das Attribut „group“ auf den Wert „grp“.

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste klickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**`checked [ state ]`**

Getriggert, nachdem der Radiobutton seinen Zustand geändert hat.

### Hinweise

Das `datasrc`-Attribut unterstützt bei Radiobuttons eine besondere Syntax:

```
datasrc="Partner.Geschlecht.WT#1"
```

Diese Notation ist erforderlich, wenn ein oder mehrere Radiobuttons an eine Liste des Objektmodells angebunden werden sollen. Im obigen Beispiel handelt es sich bei der Liste um die Wertetabelle des Geschlechts (`Partner.Geschlecht.WT`), die z.B. die Werte "männlich" (1), "weiblich" (2) und "juristisch" (3) enthalten kann. Jeder dieser Werte hat innerhalb der Wertetabelle einen eigenen Index, der hier in Klammern hinter dem jeweiligen Wert angegeben ist.

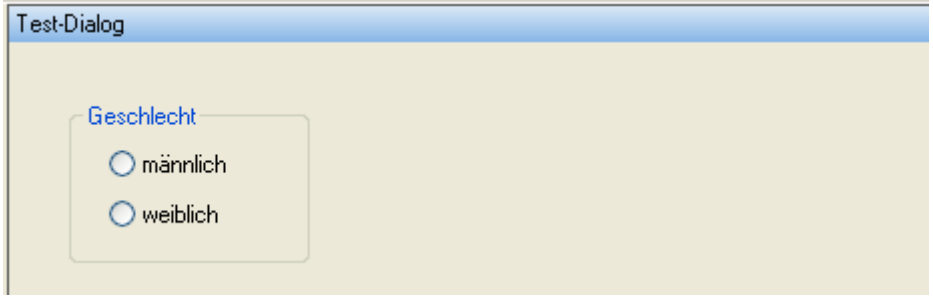

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Mit Hilfe der gezeigten Notation ist es möglich, einen Radiobutton mit genau einem dieser Werte zu assoziieren. Dabei wird automatisch der textuelle Wert (im obigen Beispiel "männlich", da Index 1) der Liste als Inhalt für das `<bw:content>`-Element des Radiobuttons verwendet (wird das <bw:content>-Element trotzdem angegeben, wird dessen expliziter Inhalt bei der Darstellung ignoriert).

### Beispieldarstellung

```
<bw:compoundbox id="CP_RBTest" tabindex="1" title="Geschlecht">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="30" eoy="30" ecx="670" ecy="200"/>
  <bw:radiobutton id="RB_1" datasrc="">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="20" ecx="200" ecy="41"/>
    <bw:content>
      <bw:t>männlich</bw:t>
    </bw:content>
  </bw:radiobutton>
  <bw:radiobutton id="RB_2" datasrc="">
    <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="45" ecx="200" ecy="66"/>
    <bw:content>
      <bw:t>weiblich</bw:t>
    </bw:content>
  </bw:radiobutton>
</bw:compound>
```

Die folgende Abbildung gibt zwei typische Radiobuttons wieder (erzeugt mit Hilfe des oben dargestellten BML-Codes). Beachten Sie bitte, dass beide `<bw:radiobutton>`-Elemente kein `tabindex`-Attribut besitzen. Damit wird eine Windows-typische Bedienfunktionalität erreicht, bei der das Umschalten zwischen den Optionen mit Hilfe der Pfeiltasten erfolgt und alle Radiobuttons einer logischen Gruppe ein Control bilden.




© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Abb. 39: Beispiel für Radiobuttons**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.24 <bw:notebook>

### Kurzbeschreibung

Ein `<bw:notebook>` ist eine Sammlung von Panes, zwischen denen mit Hilfe von Reitern (Tabs) umgeschaltet werden kann.

### Syntax

```
<bw:notebook
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      tabalign="enum_align"
      >
      <bw:pos ... />
      <bw:bookpage id="page-id">
            <bw:booktab id="tab-id" label="tab-label">
                  <bw:img ... />
            </bw:booktab>
            ... Inhalte der Notebook-Page ...
      </bw:bookpage>
</bw:notebook>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwen- |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | det und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `tabalign (E)` | (Optional) Definiert die Ausrichtung der Tabs an den Seiten des Notebooks; mögliche Werte sind: „top“, „bottom“, „left“ und „right“, entsprechend den vier Seiten des Notebooks. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `tabwidth (I)` | (Optional) Definiert die Breite der tabs innerhalb des Notebooks in Pixel. Dieses Attribut ist hilfreich, falls die Tabs alle dieselbe Breite haben sollen; andernfalls richtet sich die Breite nach der Länge des Textes bzw. optional zusätzlich des enthaltenen Images. |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:bookpage>` | Definiert die Seite innerhalb des Notebooks. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar" (falls „true" als Argument übergeben wird) bzw. auf „unsichtbar" (falls „false" als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**addEvents()**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**removeEvents()**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**selectPage( pageid)**

Macht die Seite mit der ID „pageid" zur aktuellen Seite.

**insertPage( page)**

Fügt die Seite mit dem Inhalt „page" zum Notebook hinzu.

**removePage( pageid)**

Löscht die Seite mit der ID „pageid".

**setPageContent( pageid, content)**

Setzt den Inhalt der Seite mit der ID „pageid" neu.

## Events

**clicked [ id ]**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste klickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**focuslost []**

Getriggert, wenn der Focus das Element verlässt.

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**selectionchanged [ sel ]**

Getriggert, wenn sich die Seitenauswahl geändert hat.

**selecting [ pageid ]**

Getriggert, wenn unmittelbar, nachdem der Anwender eine Seitenänderung durchgeführt hat.

**deselecting [ pageid ]**

Getriggert, wenn unmittelbar, nachdem der Anwender eine Seitenänderung durchgeführt hat.

### Hinweise

Die Inhalte innerhalb einer Tab-Seite haben immer die volle Ausdehnung der Seite; somit erübrigt sich die Angabe der Position einer Seite bzw. deren Breite/Höhe, falls diese Bestandteil einer Seite ist (ein `<bw:bookpage>`-Element benötigt somit kein `<bw:pos>`-Kind-Element).

### Beispieldarstellung

```
<bw:notebook id="NB_1">
  <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
          eox="0" eoy="0" ecx="0" ecy="0" />
  <bw:bookpage id="NP_1">
    <bw:booktab label="Hausrat">
      <bw:img src="images/angmodule.gif" />
    </bw:booktab>
    <bw:compound id="CP_Hausrat" tabindex="1">
      <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
           eox="0" eoy="0" ecx="0" ecy="0" />
      <bw:include src="Angebot_HausratHausrat.inc" />
    </bw:compound>
  </bw:bookpage>

  <bw:bookpage id="NP_2">
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    <bw:booktab label="Glas">
      <bw:img src="images/angmodule.gif" />
    </bw:booktab>
    <bw:compound id="CP_Glas" tabindex="1">
      <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
           eox="0" eoy="0" ecx="0" ecy="0" />
      <bw:include src="Angebot_HausratGlas.inc" />
    </bw:compound>
  </bw:bookpage>
</bw:notebook>
```

Das im obigen Beispiel definierte Notebook ist in der folgenden Abbildung dargestellt (der Code enthält aus Gründen der Übersichtlichkeit nicht die Definition aller Seiten; die Definition der Seiten-Inhalte wurde zudem in Dateien ausgelagert, die mit Hilfe des `<bw:include>`-Elements eingelesen werden).

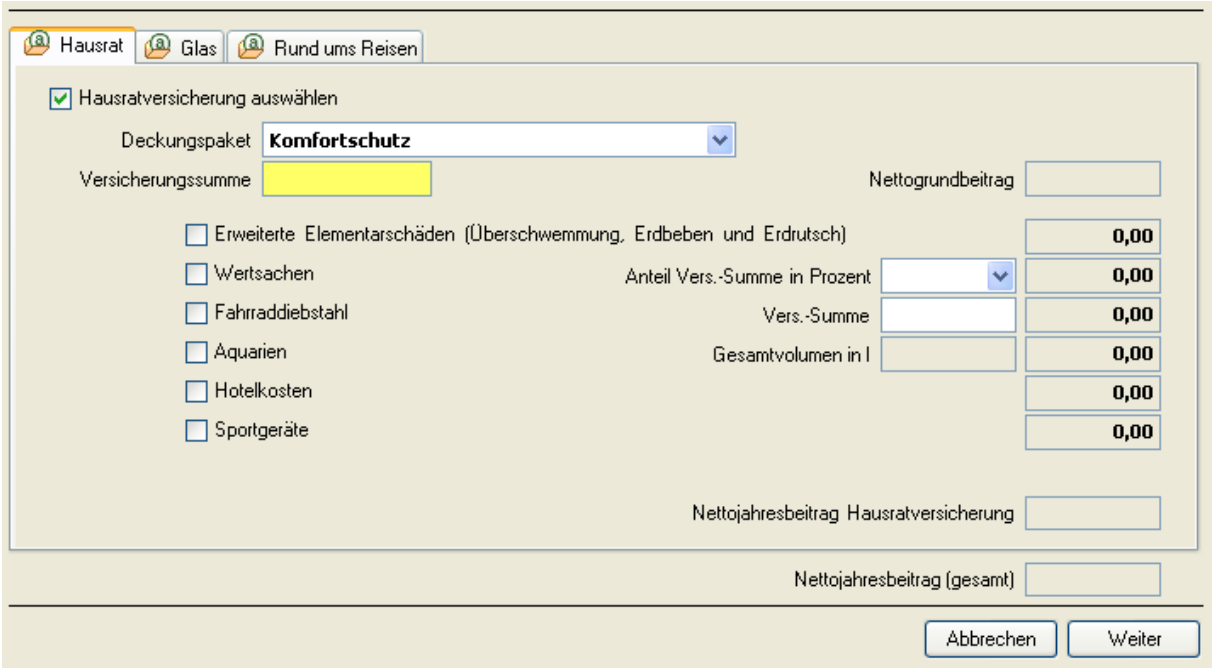



**Abb. 40: Beispiel für ein Notebook**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.24.1 <bw:bookpage>

### Kurzbeschreibung

Eine `<bw:bookpage>` ist eine Seite innerhalb eines Notebooks und kann damit ausschließlich innerhalb eines <bw:notebook>-Elements vorkommen.

ID-Konvention: „BP_“.

### Syntax

```
<bw:bookpage
      id="str_widget_id"
      >
</bw:bookpage>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `<bw:booktab>` | Definiert den Inhalt des Tabs, also der Lasche, mit deren Hilfe zu dieser Seite geschaltet werden kann. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

### API-Funktionen

Keine.

### Events

Keine.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Beispieldarstellung

Siehe Toolbar.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.24.2 <bw:booktab>

### Kurzbeschreibung

Eine `<bw:booktab>` ist eine Lasche einer Notebook-Seite.

ID-Konvention: „BPT_“.

### Syntax

```
<bw:booktab
      label="str_label"
      >
      <bw:img src="str_path" />
</bw:booktab>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `label (S)` | Der Text, der innerhalb des Tabs angezeigt werden soll. |
| `<bw:img>` | (Optional) Definiert ein optionales Icon; der Inhalt des Icons (Pfad zum Image auf dem Server) wird innerhalb des „src“-Attributs des <bw:img>-Elements angegeben. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

### API-Funktionen

Keine.

### Events

Keine.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Beispieldarstellung

Siehe Toolbar.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.25 <bw:progress>

### Kurzbeschreibung

Ein `<bw:progress>` definiert eine Fortschrittsanzeige (siehe auch Abbildung 40). Dabei kann es sich um eine klassische Fortschrittsanzeige handeln, die mit Hilfe eines grafischen Balkens einen Zustand zwischen 0 und 100% signalisiert oder um eine stetige Anzeige, bei der ein kontinuierlich laufender Balken den Fortgang einer Operation anzeigt.

ID-Konvention: „PG_“.

### Syntax

```
<bw:progress
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      min="int_minval"
      max="int_maxval"
      val="int_currval"
      step="int_step"
      barstyle="enum_style"
      datasrc="bom_element"
      >
</bw:progress>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets defi- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | niert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `barstyle (E)` | (Optional) Definiert die Art der Anzeige der Fortschrittswerte. Erlaubte Werte |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | sind „`Blocks`“, „`Continuous`“ und „`Marquee`“. Letzteres zeigt einen ständig laufenden Balken an. |
| `min (I)` | (Optional) Definiert den logischen Minimumwert. |
| `max (I)` | (Optional) Definiert den logischen Maximumwert. |
| `val (I)` | (Optional) Definiert den logischen Initialwert, mit dem der Balken angezeigt wird. |
| `step (I)` | (Optional) Definiert die logische Schrittweite (verwendet für das „performaStep“-Command) |
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer


© Copyright BISS GmbH, 2001-2012 All Rights reserved

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`setVal( val)`**

Setzt den aktuellen Wert auf „val".

**`setMin( min)`**

Setzt den Minimalwert auf „min".

**`setMax( max)`**

Setzt den Maximalwert auf „max".

**`setMinMax( min, max)`**

Setzt Minimal- und Maximalwert auf „min" bzw. „max".

**`setValue( val)`**

Setzt den aktuellen Wert auf "val".

**`setValues( min, max, val, step, barstyle)`**

Setzt alle angegeben Werte inkl. des Anzeigestils.

**`setBarStyle( style)`**

Setzt den Wert der Darstellung auf „style".

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste klickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**focusgot []**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**resized [ width, height ]**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**validate [ val ]**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**created []**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

## Beispieldarstellung

```
<bw:progress id="PB_1" tabindex="1" datasrc="Progress">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="40" eoy="20" ecx="300" ecy="40"/>
</bw:progress>
```

Die folgende Abbildung zeigt eine typische Progress-Anzeige.

**Abb. 41: Beispiele für Label-Elemente**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.26 <bw:slider>

### Kurzbeschreibung

Ein `<bw:slider>` ist ein Element, welches die Festlegung eines Wertes mit Hilfe eines Schiebereglers erlaubt.

ID-Konvention: „SL_“.

### Syntax

```
<bw:slider
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      dir="enum_dir"
      min="int_minval"
      max="int_maxval"
      val="int_currval"
      datasrc="bom_element"
      >
      <bw:content>
            Widgetcontent
      </bw:content>
</bw:slider>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets defi- |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | niert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `dir (E)` | (Optional) Definiert die Ausrichtung der progressbar. Erlaubte Werte sind „`ho-` |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| | rizontal“ und „vertical“. |
| min (I) | (Optional) Logische Untergrenze des Sliders („unter“ diesen Wert kann der Schieber nicht verschoben werden). |
| max (I) | (Optional) Logische Obergrenze („über“ diesen Werte kann der Schieber nicht verschoben werden). |
| val (I) | (Optional) Initial anzuzeigender Wert (Slider-Position). |
| datasrc (S) | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des <bw:content>-Elements ignoriert. Alternativ kann ein Label auch über ein <bw:var>-Element innerhalb des <bw:content>-Elements mit dem Objektmodell verknüpft werden. |
| <bw:style> | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs. |
| <bw:cjscript> | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| <bw:prop> | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| <bw:pos> | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| <bw:keys> | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des <bw:keys>-Tags. Siehe auch Abschnitt 9.3.2.3. |
| <bw:events> | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| <bw:tooltip> | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| <bw:contextmenu> | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| <bw:content> | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein <bw:var>-Element). |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)


© Copyright BISS GmbH, 2001-2012 All Rights reserved

B = Bool („true“ oder „false“)

## API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous“ (also fehlerhaft, falls `state = true`) oder „correct“ (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required“ (also MUSS-Element, falls `state = true`) oder „normal“ (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar“ (falls „true“ als Argument übergeben wird) bzw. auf „unsichtbar“ (falls „false“ als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

**`setRange( min, max)`**

Setzt Minmal- und Maximalwert auf „min“ bzw. „max“.

**`setValue( val)`**

Setzt den aktuellen Wert auf „val“.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle“ des Elements) mit der linken Maustaste klickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id“ definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert“. Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**pcontentchanged []**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**pvstatechanged []**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

**scroll [ val ]**

Getriggert, wenn der Slider bewegt wird. Da dieser Event häufig triggert, wenn der Slider bewegt wird, sollte er möglichst nur zusammen mit Skript-Funktionen verwendet werden.

**valuechanged [ val ]**

Getriggert, nachdem sich der Wert des Sliders geändert hat.

**pvaluechanged [ val ]**

Getriggert, nachdem der Wert des Slider vom Prozess oder vom Skript geändert worden ist.

### Beispieldarstellung

```
<bw:slider id="SL_1" dir="horizontal">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
    eox="120" eoy="20" ecx="500" ecy="40" />
</bw:slider>

...

<bw:slider id="SL_2" dir="horizontal" min="0" max="200"
  dragmin="20" dragmax="150" val="80" border="Flat">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
    eox="120" eoy="80" ecx="500" ecy="108" />
  <bw:thumbimg src="images/sliderbutton.png" />
</bw:slider>
```

Die folgende Abbildung enthält zwei Beispiele für ein Slider-Element. Das obere Slider-Element zeigt die Standard-Darstellung unter Windows 7, das untere Slider-Element enthält einen individuell definierten (Image) Schiebe-Button.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Abb. 42: Beispiel von zwei horizontalen Slider-Elementen**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.27 <bw:picturebox>

### Kurzbeschreibung

Eine `<bw:picturebox>` ist ein aktives grafisches Element. Visuell entspricht es einem `<bw:img>`-Element, verfügt jedoch im Ggs. zu diesem über die volle Funktionalität eines Controls (Reaktion auf Maus- oder Tastatur-Events, vollständiges API eines Controls etc.).

Genauer gesagt handelt es sich bei der `<bw:picturebox>` um ein Container-Element, in das mit Hilfe des Kind-Elements `<bw:img>` ein beliebiges Image gesetzt werden kann.

ID-Konvention: „PB_“.

### Syntax

```
<bw:picturebox
      id="str_widget_id"
      styleref="str_style_reference"
      class="str_styleclass"
      tabindex="int_tabindex"
      state="enum_state"
      enabled="bool"
      visible="bool"
      border="enum_bordertype"
      sizemode="enum_scaling"
      >
      <bw:img src="image_src">
</bw:picturebox>
```

### Parameter

Der Wert in Klammern hinter dem Attributnamen definiert den Typ des Attributs (siehe Legende unterhalb der Tabelle).

| Parameter | Beschreibung |
|---|---|
| `id (S)` | Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID eine `div`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Siehe auch *B-Wise* Namenskonventionen im Anhang. |
| `styleref (S)` | (Optional)Referenz auf eine Style-Definition, die außerhalb dieses Widgets definiert ist (als Style-Template); siehe hierzu den Abschnitt 9.9 „Styles“. |

© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `class (S)` | (Optional) Das class-Attribut wird häufig als „Style-ID“ bezeichnet und verwendet und eine externe Style-Template-Definition zu verwenden. Weitere Informationen und auch Beispiele hierzu sind im Abschnitt 9.9 „Styles“ aufgeführt. |
| `state (E)` | (Optional) Der initiale Status des Widgets; `state` kann dabei die folgenden Werte annehmen:<br>`initial` = Initial (die Darstellungswerte des Normalfalls, d.h. gültig, bedienbar und weder „`required`“ noch „`erroneous`“); entspricht auch dem „Zurücksetzen auf den Normalzustand<br>`required` = Darstellung als MUSS-Element<br>`erroneous` = Darstellung als Element mit logisch fehlerhaftem Inhalt<br>Siehe hierzu auch den Abschnitt 9.3.1.<br>`disabled` = Darstellung in inaktiver Form; das Element kann in diesem Zustand nicht bedient werden, dessen Inhalte sind ungültig bzw. irrelevant<br>`readonly` = Darstellung in nicht bedienbarer Form, die Inhalte sind jedoch gültig bzw. relevant<br>Standard-Einstellung ist `state="initial"`. |
| `tabindex (I)` | (Optional) Eine Integer-Zahl, welche die Fokus-Reihenfolge beim Drücken der TAB-Taste bestimmt. Die Zahlenfolge innerhalb eines Dialogs muss dabei nicht zwangsläufig kontinuierlich sein; der Fokus springt dabei jeweils zum Widget mit der nächsthöheren Zahl (also z.B. von der 4 zur 8, falls die Werte 5, 6 und 7 nicht existieren). Jedes Widget mit einem `tabindex` ist fokussierbar, also auch ein Label; das Weglassen des `tabindex`-Attributs hat automatisch zur Folge, dass das betreffende Widget nicht fokussierbar ist. |
| `visible (B)` | (Optional) Stellt das Element dar (visible=“`true`“) oder nicht (visible=“`false`“); Standard-Einstellung ist visible=“`true`“. |
| `rolocked (B)` | (Optional) Wenn „true“, verhindert Hittesting und die Erkennung von Events im „readonly“-Zustand; damit kann z.B. verhindert werden, dass ein Button im „readonly“-Zustand auf „mouse-over“-Ereignisse reagiert |
| `busyactive (B)` | (Optional) Wenn „true“, erlaubt es dem Element auch dann aktiv zu sein, wenn sich das zugehörige Portlet im „active-lock“-Zustand befindet, also temporär gesperrt ist. |
| `border (E)` | (Optional) Definiert einen sichtbaren Rahmen (border) um das Element. Mögliche Werte für dieses Attribut sind:<br>None = Keinen Rahmen zeichnen<br>FixedSingle = Einfachen Rahmen um das Element zeichnen<br>Fixed3D = Rahmen mit 3D-Effekt („inset“) um das Element zeichnen<br>Der Rahmen wird innerhalb der Boundaries des Elements gezeichnet. |
| `borderref (S)` | (Optional) Verweis auf die Definition eines Rahmens; diese Definition wird mit Hilfe des „<bw:tborder>“-Elements durchgeführt (siehe auch Abschnitt 9.9.4). |
| `sizemode (E)` | (Optional) Definiert die Art der Bilddarstellung. Erlaubte Werte sind „`Normal`“, „`StretchImage`“, „`AutoSize`“, „`CenterImage`“ und „`Zoom`“. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `datasrc (S)` | (Optional) Definiert eine String-ID, die das Attribut des Objektmodells repräsentiert, dessen Inhalt als Content des Labels angezeigt werden soll. Wird das datasrc-Attribut verwendet, wird der Inhalt des `<bw:content>`-Elements ignoriert. Alternativ kann ein Label auch über ein `<bw:var>`-Element innerhalb des `<bw:content>`-Elements mit dem Objektmodell verknüpft werden. |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Widgets verwendet werden. Angaben innerhalb des style-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des `class`-Attributs. |
| `<bw:cjscript>` | (Optional) Ermöglicht die Definition eines dynamischen, clientseitig ausgewerteten Skripts mit Bezug auf das aktuelle Element. Siehe hierzu auch den Abschnitt 9.13. |
| `<bw:prop>` | (Optional) Erlaubt die Definition zusätzlicher Element-Attribute. Das „<bw:prop>“-Element greift direkt auf die Basisplattform (z.B. .NET) zu. Bei Attributen, die sowohl native als auch mit Hilfe der in BML definierten Attribute oder „inner-tags“ gesetzt werden können (z.B. background-color) haben die in BML definierte Werte Vorrang. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:keys>` | (Optional) Definiert eine Tastatur-Zuordnungsdatei sowie ggf. auch einzelne Tastatur-Zuordnungen (siehe auch allgemeine Beschreibung des `<bw:keys>`-Tags. Siehe auch Abschnitt 9.3.2.3. |
| `<bw:events>` | (Optional) Definiert eine Event-Zuordnungsdatei sowie ggf. einzelne Event-Handler für bestimmte Events. Siehe auch Abschnitt 9.3.2.4. |
| `<bw:tooltip>` | (Optional) Definiert einen Hilfetext, der als „Tooltip“ innerhalb eines Rechteck-Bereichs beim überfahren des Elements mit der Maus angezeigt wird. Siehe auch Abschnitt 9.3.2.5. |
| `<bw:contextmenu>` | (Optional) Definiert für das Widget ein Kontext-Menü. Details hierzu siehe Abschnitt 9.8.18 Menüs. |
| `<bw:content>` | Enthält den eigentlichen Inhalt des Widgets. Beim Label kann hier sowohl konstanter Text als auch ein Ausdruck steht, der zu einem String (Text) evaluiert (z.B. ein `<bw:var>`-Element). |
| `<bw:img>` | Definiert mit Hilfe des „src“-Attributs den Pfad zum Image auf dem Server. |

**Legende der Attribut-Typen:**

S = String

I = Integer

E = Enumeration (Aufzählungstyp, d.h. eine der erlaubten Optionen)

B = Bool („true“ oder „false“)

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### API-Funktionen

**`setContent( newcontent )`**

Setzt den Inhalt des Elements auf den als Argument `newcontent` übergebenen Wert (entspricht dem Setzen des `<bw:val>`-Knotens innerhalb des Label-Tags).

**`setErroneous( state )`**

Setzt den Zustand des Elements auf den logischen Wert „erroneous" (also fehlerhaft, falls `state = true`) oder „correct" (also nicht fehlerhaft, falls `state = false`).

**`setRequired( state )`**

Setzt den Zustand des Elements auf den logischen Wert „required" (also MUSS-Element, falls `state = true`) oder „normal" (also kein MUSS-Feld, falls `state = false`).

**`setState( state )`**

Setzt den Zustand des Elements auf den logischen Wert der mit `state` angegeben ist; dabei kann es sich um einen der erlaubten Stati handeln (s.o.).

**`setFocus()`**

Setzt den Focus auf das Label-Element.

**`setVisible()`**

Setzt den Zustand des Elements auf den logischen Wert „sichtbar" (falls „true" als Argument übergeben wird) bzw. auf „unsichtbar" (falls „false" als Argument übergeben wird).

**`setCtxMenu()`**

Setzt für das betreffende Element ein Kontext-Menü. Das Menü selbst kann als Referenz oder unmittelbar als Parameter dieser Methode angegeben werden.

**`setTooltip( item, tooltip )`**

Setzt für das Treeitem `item` den Tooltip (die Bubblehelp) auf den als `tooltip` übergebenen Wert.

**`setStyle( attrname, attrval )`**

Setzt den als Argument `attrname` übergebenen Style (z.B. `font-size`) auf den als Argument übergebenen Wert `attrval`; Beispiel: `setStyle("font-size", "10pt")`. Ersetzt den bisherigen Style.

**`setStyleRef( styleref )`**

Setzt den als Argument `styleref` übergebenen Style. Ersetzt den bisherigen Style.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**`addEvents()`**

Fügt zu den bisher definierten Events die als Argument übergebenen Events hinzu. Wird ein Event mit unterschiedlichen Aktionen mehrfach hinzugefügt, werden die entsprechenden Aktionen als Liste in der definierten Reihenfolge abgearbeitet.

**`removeEvents()`**

Löscht die als Argument übergebenen Events und ihre Aktionen aus der Eventliste.

## Events

**`clicked [ id ]`**

Getriggert, wenn der Benutzer auf das Element (besser auf das sog. „bounding rectangle" des Elements) mit der linken Maustaste klickt. Übergeben wird die ID (`id`) des Elements, die mit Hilfe des Attributs „id" definiert worden ist.

**`focuslost []`**

Getriggert, wenn der Focus das Element verlässt.

**`focusgot []`**

Getriggert, wenn der Focus auf das Element gesetzt wird.

**`resized [ width, height ]`**

Getriggert, nachdem das Element seine Größe geändert hat. Als Parameter werden die neue Größe (Breite und Höhe in Pixel) mitgegeben.

**`validate [ val ]`**

Getriggert, wenn der Benutzer den Inhalt des Elements „validiert". Die Validierung eines Elements ist vom jeweiligen Element-Typ abhängig. Dem Event wird beim Triggern der aktuelle Wert des Elements mitgegeben.

**`created []`**

Getriggert, wenn unmittelbar, nachdem das Element innerhalb des clientseitigen DOM erzeugt worden ist.

**`pcontentchanged []`**

Getriggert, nachdem der Server (per setContent) den Inhalt des Elements geändert hat. Dieser Event wird *nicht* getriggert, wenn sich der Inhalt clientseitig ändert (z.B. durch Benutzereingaben bei einem Entryfield).

**`pvstatechanged []`**

Getriggert, nachdem der Server den Zustand (z.B. readonly) geändert hat.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Beispieldarstellung

```
<bw:picturebox id="PB_1" border="FixedSingle" sizemode="CenterImage">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="50" eoy="50" ecx="200" ecy="200"/>
  <bw:img src="images/AnalyzeWorld.gif" />
</bw:picturebox>
```

Die folgende Abbildung enthält die Darstellungsmodi einer Pixturebox mit einem Image.

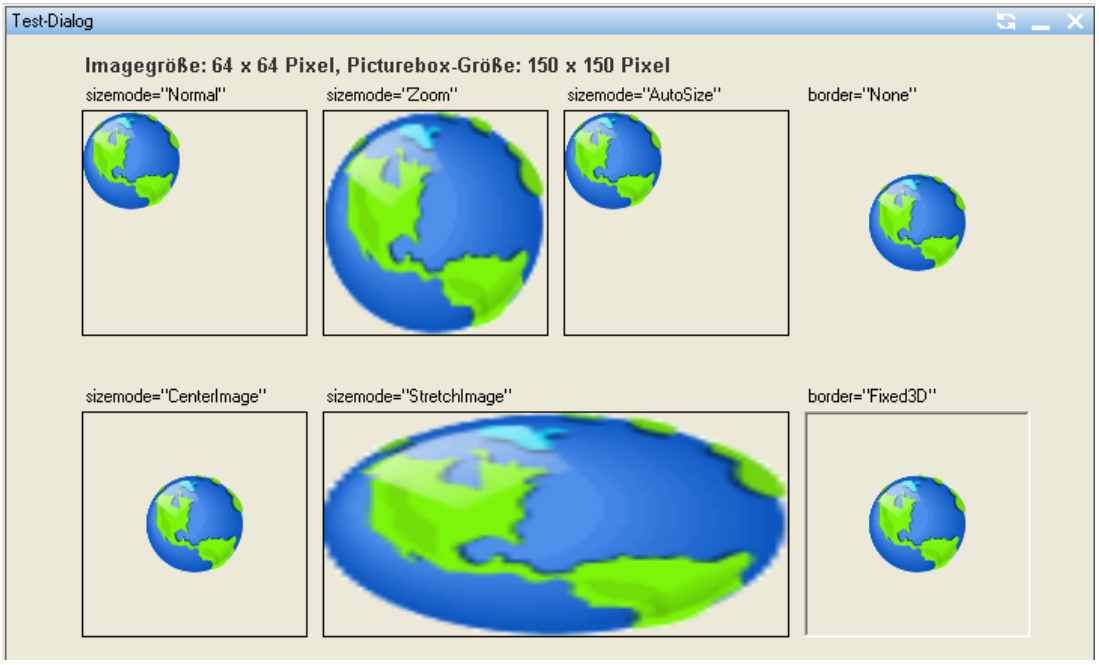



**Abb. 43: Beispiele für Picturebox-Elemente**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.28 Deco-Elemente

**Kurzbeschreibung**

Deco-Elemente (für „Decorations"-Elemente) sind grafische Elemente, die keine Funktion besitzen. Wie ihr Name schon sagt, stellen sie reine Dekorationen innerhalb eines Dialogs dar.

Im Ggs. zu aktiven Controls sind sie „leichtgewichtig", d.h. beanspruchen zur Laufzeit weniger Overhead und damit weniger Speicherplatz. Insbesondere, wenn viele, statische Elemente eines definierten Typs benötigt werden, sind Deco-Elemente zu bevorzugen.

**Hinweis:** Dekorationselemente sind statisch, d.h. verfügen weder über ein API noch triggern sie irgendwelche Ereignisse (Events).

Folgende Deco-Elemente stehen zur Verfügung und sind in diesem Abschnitt beschrieben:

```
<bw:line>
<bw:rect>
<bw:circle>
<bw:groupbox>
<bw:img>
<bw:text>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.28.1 <bw:line>

### Kurzbeschreibung

Das `<bw:line>`-Element ist eines von sechs Dekorations-Elementen. Es stellt eine gemäß den Styleangaben präsentierte Linie dar. Die Linie wird zwischen den beiden Ecken des Bounding Rectangles aufgespannt.

**Hinweis:** Dekorationselemente sind statisch, d.h. verfügen nicht über ein API bzw. Triggern auch keine Ereignisse (Events).

ID-Konvention: „DL_“.

### Syntax

```
<bw:line
   id="widget-id"
   class="style-class"
   >
</bw:line>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID einer `<bw:div>`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Diese ID wird nur benötigt, wenn das Tooltip außerhalb eines Widgets definiert und über eine Referenz angesprochen wird. Wird das Tooltip innerhalb eines Widgets definiert, kann die ID entfallen. Es ist dann allerdings nicht möglich, das Tooltip programmgesteuert anzusprechen. |
| `zorder (I)` | (Optional) Definiert die sog. Z-Order (also die relative Position des Element in Bezug auf andere Element in Z-Richtung). Die Zahl bedeutet, dass das Element über allen Elementen liegt deren eigene Z-Order-Zahl kleiner ist. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) der Linie verwendet werden. Angaben innerhalb des `<bw:style>`-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs. |

### Beispieldarstellung

```
<bw:line id="DL_99">
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
           eox="5" eoy="20" ecx="-5" ecy="20"/>
</bw:line>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.28.2 <bw:rect>

### Kurzbeschreibung

Das `<bw:rect>`-Element ist eines von sechs Dekorations-Elementen. Es stellt ein gemäß den Styleangaben präsentiertes Rechteck dar. Optional kann das Rechteck gefüllt sein (opaque) oder transparent (Default-Einstellung).

**Hinweis:** Dekorationselemente sind statisch, d.h. verfügen nicht über ein API bzw. Triggern auch keine Ereignisse (Events).

ID-Konvention: „DR_“.

### Syntax

```
<bw:rect
   id="widget-id"
   fill="bool"
   zorder="int_order"
   >
</bw:rect>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID einer `<bw:div>`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Diese ID wird nur benötigt, wenn das Tooltip außerhalb eines Widgets definiert und über eine Referenz angesprochen wird. Wird das Tooltip innerhalb eines Widgets definiert, kann die ID entfallen. Es ist dann allerdings nicht möglich, das Tooltip programmgesteuert anzusprechen. |
| `fill (B)` | (Optional) Definiert (falls gleich „`true`“), dass das Rechteck mit einer definierbaren Farbe gefüllt werden soll; die Farbe kann als Inhalt des `<bw:style>`- bzw. darin des `<bw:backgroundcolor>`-Elements angegeben werden (siehe auch Beispiel unten). |
| `zorder (I)` | (Optional) Definiert die sog. Z-Order (also die relative Position des Element in Bezug auf andere Element in Z-Richtung). Die Zahl bedeutet, dass das Element über allen Elementen liegt deren eigene Z-Order-Zahl kleiner ist. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Rechtecks verwendet werden. Angaben innerhalb des <bw:style>-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs.

### Beispieldarstellung

```
<bw:rect id="DR_1" fill="true" zorder="22">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="10" eoy="90" ecx="10" ecy="110" />
  <bw:style>
    <bw:color>#7f9db9</bw:color>
    <bw:backgroundcolor>#e9e9e9</bw:backgroundcolor>
  </bw:style>
</bw:rect>
```

Die folgende Abbildung zeigt neben einer Ellipse und einem Kreis ein gefülltes Rechteck.

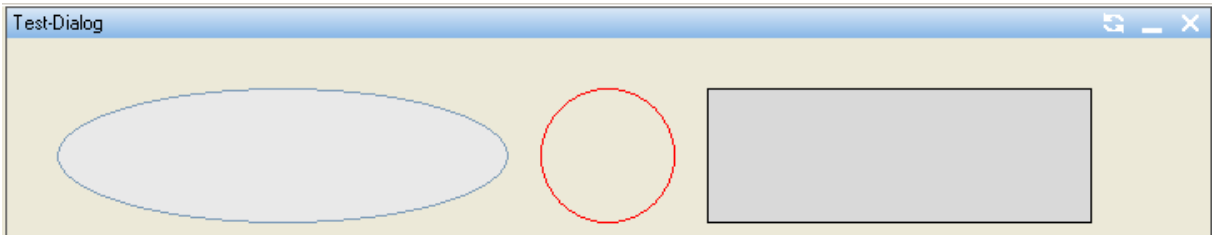



**Abb. 44: Beispiel für ein Rechteck (rechts neben Ellipse und Kreis)**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.28.3 <bw:groupbox>

### Kurzbeschreibung

Das `<bw:groupbox>`-Element ist eines von sechs Dekorations-Elementen. Es stellt ein gemäß den Styleangaben präsentiertes Rechteck dar, das optional einen Text haben kann, der innerhalb der oberen Randlinie angezeigt wird (siehe auch Abbildung weiter unten). Die Groupbox hat auf den meisten Plattformen runde Ecken, lässt sich allerdings nicht „stylen", besitzt also eine definierte Farbe und ist immer transparent.

**Hinweis:** Dekorationselemente sind statisch, d.h. verfügen nicht über ein API bzw. Triggern auch keine Ereignisse (Events).

ID-Konvention: „DB_".

### Syntax

```
<bw:groupbox
  id="Str_widget_id"
  title="text"
/>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID einer `<bw:div>`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Diese ID wird nur benötigt, wenn das Tooltip außerhalb eines Widgets definiert und über eine Referenz angesprochen wird. Wird das Tooltip innerhalb eines Widgets definiert, kann die ID entfallen. Es ist dann allerdings nicht möglich, das Tooltip programmgesteuert anzusprechen. |
| `title (S)` | (Optional) Der Text, der innerhalb der oberen Begrenzungslinie eingefügt werden soll. |
| `<bw:pos>` | Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Rechtecks verwendet werden. Angaben innerhalb des `<bw:style>`-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs. |

### Beispieldarstellung

```
<bw:groupbox title="Beispiel-Text">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="20" eoy="10" ecx="220" ecy="130"/>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
</bw:groupbox>
```

Die folgende Abbildung stellt eine typische Group-Box dar.

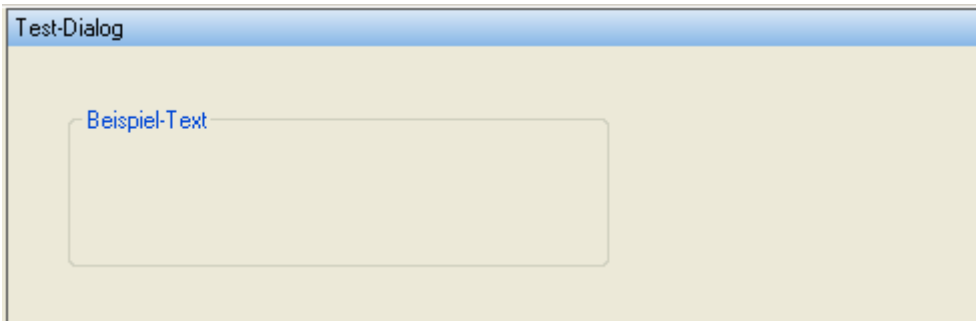



**Abb. 45: Beispiel einer Group-Box**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.8.28.4 <bw:circle>

### Kurzbeschreibung

Das `<bw:circle>`-Element ist eines von sechs Dekorations-Elementen. Es stellt einen gemäß den Styleangaben präsentierten Kreis dar.

**Hinweis:** Dekorationselemente sind statisch, d.h. verfügen nicht über ein API bzw. Triggern auch keine Ereignisse (Events).

ID-Konvention: „DC_“.

### Syntax

```
<bw:circle
   id="widget-id"
   fill="bool"
   zorder="int_order"
   >
</bw:circle>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID einer `<bw:div>`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Diese ID wird nur benötigt, wenn das Tooltip außerhalb eines Widgets definiert und über eine Referenz angesprochen wird. Wird das Tooltip innerhalb eines Widgets definiert, kann die ID entfallen. Es ist dann allerdings nicht möglich, das Tooltip programmgesteuert anzusprechen. |
| `fill (B)` | (Optional) Definiert (falls gleich „`true`“), dass der Kreis bzw. die Ellipse mit einer definierbaren Farbe gefüllt werden soll; die Farbe kann als Inhalt des `<bw:style>`- bzw. darin des `<bw:backgroundcolor>`-Elements angegeben werden (siehe auch Beispiel unten). |
| `zorder (I)` | (Optional) Definiert die sog. Z-Order (also die relative Position des Elements in Bezug auf andere Element in Z-Richtung). Die Zahl bedeutet, dass das Element über allen Elementen liegt deren eigene Z-Order-Zahl kleiner ist. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Rechtecks verwendet werden. Angaben innerhalb des <bw:style>-Tags überschreiben |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

(oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs.

### Beispieldarstellung

```
<bw:circle id="DR_1" fill="true" zorder="22">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="10" eoy="90" ecx="10" ecy="110" />
  <bw:style>
    <bw:color>#7f9db9</bw:color>
    <bw:backgroundcolor>#e9e9e9</bw:backgroundcolor>
  </bw:style>
</bw:circle>
```

Die folgende Abbildung zeigt neben einem Rechteck zusätzlich eine Ellipse sowie einen Kreis.

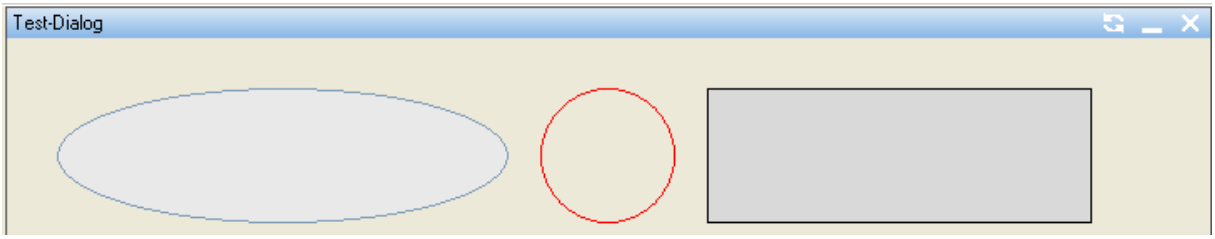



**Abb. 46: Beispiel eines Kreises/einer Ellipse**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.28.5 <bw:img>

### Kurzbeschreibung

Das `<bw:img>`-Element ist eines von sechs Dekorations-Elementen. Es stellt ein gemäß den Styleangaben präsentiertes Bild (Image) dar.

**Hinweis:** Dekorationselemente sind statisch, d.h. verfügen nicht über ein API bzw. Triggern auch keine Ereignisse (Events).

ID-Konvention: „DI_".

### Syntax

```
<bw:img
   id="widget-id"
   src="image-source"
   mode="enum_scale-flag"
   align="enum-align"
   zorder="int-zorder"
   >
</bw:img>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID einer `<bw:div>`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Diese ID wird nur benötigt, wenn das Tooltip außerhalb eines Widgets definiert und über eine Referenz angesprochen wird. Wird das Tooltip innerhalb eines Widgets definiert, kann die ID entfallen. Es ist dann allerdings nicht möglich, das Tooltip programmgesteuert anzusprechen. |
| `mode (E)` | (Optional) Definiert die Art der Darstellung des Image; mögliche Werte sind: „unscaled", „scaled", „centered", „cropped", „aligned" |
| `align (E)` | (Optional) Definiert die Ausrichtung des Image innerhalb der Grenzen des Image-Elements, falls „mode" den Wert „Aligned" hat; mögliche Werte sind eine Kombination aus „top", „middle", „bottom" sowie „left", „center", „right". |
| `src (S)` | Definiert den Pfad zum Image auf dem Server. |
| `zorder (I)` | (Optional) Definiert die sog. Z-Order (also die relative Position des Elements in Bezug auf andere Element in Z-Richtung). Die Zahl bedeutet, dass das Element über allen Elementen liegt deren eigene Z-Order-Zahl kleiner ist. |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Rechtecks verwendet werden. Angaben innerhalb des `<bw:style>`-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs. |

### Hinweise

Ein Image-Element wird immer in seiner definierten Größe dargestellt und innerhalb der durch das `<bw:pos>`-Element definierten Box zentriert. Grundsätzlich kann somit anstatt einer Box auch ein Punkt definiert werden. An diesem Punkt wird dann das Image zentriert dargestellt.

### Beispieldarstellung

```
<bw:img id="DI_1" src="images/AngebotLogo.gif" zorder="15">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
           eox="10" eoy="10" ecx="50" ecy="50"/>
</bw:img>
```

Die folgende Abbildung zeigt ein einfaches Image-Element. Beachten Sie, dass es optisch nicht von einer Picturebox ohne Rahmen zu unterscheiden ist. Das Image-Element ist jedoch passiv, d.h. reagiert weder auf Maus-Events, noch auf Tastatur-Eingabe oder verfügt über ein eigenes API.

**Abb. 47: Beispiel eines Image-Elements**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.8.28.6 <bw:text>

### Kurzbeschreibung

Das `<bw:text>`-Element ist eines von sechs Dekorations-Elementen. Es stellt einen gemäß den Styleangaben präsentierten Text dar. In vielerlei Hinsicht entspricht es dem <bw:label>-Element, ist jedoch statisch, d.h. kann nicht über ein API erreicht werden. Im Ggs. zum `<bw:label>` ist es jedoch „leichtgewichtiger", beansprucht also zur Laufzeit weniger Speicherplatz (Overhead).

**Hinweis:** Dekorationselemente sind statisch, d.h. verfügen nicht über ein API bzw. Triggern auch keine Ereignisse (Events).

ID-Konvention: „DT_".

### Syntax

```
<bw:text
   id="widget-id"
   align="enum-align"
   zorder="int-zorder"
   >
</bw:text>
```

### Parameter

| Parameter | Beschreibung |
|---|---|
| `id (S)` | (Optional) Eine innerhalb der BML-Seite eindeutige ID des Widgets; diese ID mapped beim Browser auf die ID einer `<bw:div>`-Section und muss daher auch eindeutig in Bezug auf alle anderen IDs der BML-Seite sein. Diese ID wird nur benötigt, wenn das Tooltip außerhalb eines Widgets definiert und über eine Referenz angesprochen wird. Wird das Tooltip innerhalb eines Widgets definiert, kann die ID entfallen. Es ist dann allerdings nicht möglich, das Tooltip programmgesteuert anzusprechen. |
| `align (E)` | (Optional) Definiert die Ausrichtung des Image innerhalb der Grenzen des Image-Elements, falls „mode" den Wert „Aligned" hat; mögliche Werte sind eine Kombination aus „top", „middle", „bottom" sowie „left", „center", „right". |
| `zorder (I)` | (Optional) Definiert die sog. Z-Order (also die relative Position des Elements in Bezug auf andere Element in Z-Richtung). Die Zahl bedeutet, dass das Element über allen Elementen liegt deren eigene Z-Order-Zahl kleiner ist. |
| `<bw:pos>` | (Optional) Ein Tag, welches die Positionsangaben enthält (siehe hierzu Abschnitt 9.3.2.2). |


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `<bw:style>` | (Optional) Ein Format-Tag, dessen Angaben für die Darstellung (das Layout) des Rechtecks verwendet werden. Angaben innerhalb des `<bw:style>`-Tags überschreiben (oder ergänzen) ggf. vorhandene Angaben innerhalb des class-Attributs. |

### Beispieldarstellung

```
<bw:text>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
          eox="10" eoy="10" ecx="50" ecy="50"/>
</bw:text>
```

Die folgende Abbildung zeigt ein einfaches, statisches Text-Element. Beachten Sie, dass es optisch nicht von einem `<bw:label>` zu unterscheiden ist. Das Text-Element ist rein passiv, d.h. reagiert weder auf Maus-Events, noch auf Tastatur-Eingabe oder verfügt über ein eigenes API.

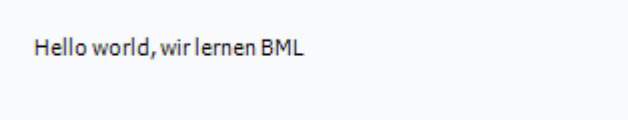



**Abb. 48: Beispiel eines Text-Elements**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.9 Styles

Das Aussehen von BML-Dialogen wird wesentlich mit Hilfe von Styles definiert. Styles können sowohl unmittelbar beim Element angegeben werden (sog. „local styles“) oder sie können innerhalb einer separaten Datei als benamte Schablonen (Templates) hinterlegt und mit Hilfe von Regeln an Elemente gebunden werden.

Regeln werden mit Hilfe von Pfadausdrücken definiert, so dass vielfältige Möglichkeiten zur Anbindung von Styles an Elemente bestehen. Bei Verwendung der Zielplattform „Browser“ ist jedoch zu beachten, dass diese Regeln auf CSS abgebildet werden müssen, so dass lediglich ein Subset der verfügbaren Regeln verwendet werden kann.

Template-Styles können global für eine Anwendung oder lokal für ein Portlet definiert werden. Kommt es dabei zu Namenskonflikten zwischen der globalen sowie der Portlet-Definition, greifen entsprechende Prioritätsregeln. Dabei wird folgende Reihenfolge verwendet (angefangen mit der niedrigsten Priorität):

- Portal-Styles (bzw. globale Styles)
- Portlet-Styles
- Lokale Styles (Element-Definitionen)

D.h., eine lokale Style-Definition überschreibt die entsprechende Portlet-Styledefinition und diese wiederum die Portal-Style Definition.

Folgende Style-Tags stehen in der Version 4.03 zur Verfügung:

### 9.9.1 <bw:style> bzw. <bw:tstyle>

Das `<bw:style>`- bzw. das `<bw:tstyle>`-Tag definiert einen lokalen Style bzw. eine benamte Schablone (Template) sowie im Falle des Templates die entsprechende Regel, auf welche Dialog-Elemente dieses Template anzuwenden ist. Die einzelnen Layout-Definitionen sind innerhalb des `<bw:style>`- bzw. innerhalb des `<bw:tstyle>`-Tags mit Hilfe der entsprechenden spezifischen Style-Tags definiert (siehe den Abschnitt 9.9.2 „Style-Attribute“).

Das `<bw:style>`-Tag verfügt über keine Attribute.

Das `<bw:tstyle>`-Tag verfügt über zwei Attribute:

| Attribut | Beschreibung |
|---|---|


© Copyright BISS GmbH, 2001-2012 All Rights reserved

| | |
|---|---|
| `id` | (Optional) Definiert einen eindeutigen Namen des Templates. Diese ID kann innerhalb des `styleref`-Attributs eines Elements verwendet werden. Dies definiert eine alternative Assoziationsmöglichkeit zur unten genannten Assoziation mit Hilfe von `select`-Regeln. |
| `select` | Definiert eine Regel, auf welche Dialog-Elemente dieses Template anzuwenden ist. Diese Definition erfolgt mit Hilfe eines Pfadausdrucks der Form „key=value“ (siehe auch Beispiele weiter unten). |

### 9.9.2 Style-Attribute

Style-Attribute bestimmen die eigentlichen Inhalte des `<bw:style>`- bzw. `<bw:tstyle>`-Tags und damit das Layout des Elements bzw. der Elemente. Sie sind mit ihren Namen unmittelbar als Elemente (Tags) definiert. Die in der folgenden Liste angegebenen Namen können daher direkt als Elemente verwendet werden. Bei der Schreibweise der Namen wird grundsätzlich die Kleinschreibung verwendet. Der Wert des Styles wird als Wert des Elements angegeben.

Folgende Style-Attribute stehen in der Version 4.03 zur Verfügung bzw. sind für die Version 5.0 geplant (in alphabetischer Reihenfolge, in Klammern dahinter die Elementschreibweise; die Namen wurden in Anlehnung an CSS definiert, so dass sich für einen mit CSS erfahrenen Entwickler/Designer eine einfache Entsprechung ergibt):

- backgroundcolor (`<bw:backgroundcolor>`)
- border (`<bw:border>`)
- borderbottom (`<bw:borderbottom>`)
- borderleft (`<bw:borderleft>`)
- borderright (`<bw:borderright>`)
- bordertop (`<bw:bordertop>`)
- color (`<bw:color>`)
- font (`<bw:font>`)
- fontfamily (`<bw:fontfamily>`)
- fontsize (`<bw:fontsize>`)
- fontstyle (`<bw:fontstyle>`)
- fontweight (`<bw:fontweight>`) - **5.0**
- height (`<bw:height>`) - **5.0**
- horizontalalign (`<bw:horizontalalign>`) - **5.0**
- linespacing (`<bw:linespacing>`) - **5.0**
- padding (`<bw:padding>`)
- paddingbottom (`<bw:paddingbottom>`) - **5.0**
- paddingleft (`<bw:paddingleft>`) - **5.0**
- paddingright (`<bw:paddingright>`) - **5.0**

© Copyright BISS GmbH, 2001-2012 All Rights reserved

- paddingtop (<bw:paddingtop>) - **5.0**
- verticalalign (<bw:verticalalign>) - **5.0**
- textdecoration (<bw:textdecoration>) - **5.0**
- width (<bw:width>)

**Beispiel:**

```
<bw:group xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:var name="font1" expr="Verdana,Arial" />
  <bw:var name="bckgr" expr="#D9D9D9" />

  <bw:tstyle id="s001" select="@class='eingabe'">
    <bw:fontface><bw:var expr="$font1" /></bw:fontface>
    <bw:fontsize>10pt</bw:fontsize>
    <bw:color>black</bw:color>
    <bw:backgroundcolor><bw:var expr="{$bckgr}" /></bw:backgroundcolor>
  </bw:tstyle>

  <bw:tstyle id="s002" select="@id='pane2'">
    <bw:backgroundcolor>white</bw:backgroundcolor>
  </bw:tstyle>

</bw:group>
```

### 9.9.3 Anwendung

Der folgende Abschnitt beschreibt kurz die Art und Weise, wie Style-Dateien genutzt werden können bzw. gibt Hinweise zur internen Arbeitsweise des Transformationsvorgangs bezüglich der Styles.

**Lokale Styles** werden innerhalb der einzelnen Widget-Elemente verwendet. Beispiel (die Style-Definition ist durch Fettdruck hervorgehoben):

```
<bw:label id="LA_Kapitel" align="TopLeft" class="lab">
  <bw:style>
    <bw:fontstyle>Bold</bw:fontstyle>
  </bw:style>
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
    eox="18" eoy="15" ecx="118" ecy="35" />
  <bw:content>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    <bw:t>Kapitelauswahl</bw:t>
  </bw:content>
</bw:label>
```

Im obigen Beispiel wird die Schriftart des Labels auf Fettdruck gesetzt.

**Template-Styles** können auf zwei Ebenen angewendet werden:

- Portal-Ebene
- Portlet-Ebene

Durch die Strukturierung der Dialog-Dateien gibt es mehrere Möglichkeiten, Portal-Styles zu definieren (die entsprechenden Includes sind durch Fettdruck hervorgehoben).

Innerhalb der zentralen Datei für den SmartClient (`BWisePortalBasePageSC.bml`)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bw:group xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:include src="genericstyles.xml" />

  <bw:include src="BWisePortalBasePage.inc"/>

</bw:group>
```

Innerhalb der jeweiligen Portal-Datei (z.B. `BWiseInfoPortalBasePage.inc`)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bw:group
  xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:include src="portalstyles.xml" />

  <bw:!-- Portlet-Container -->
  <bw:portal id ="BWisePortalBasePage" name="BWisePortalBasePage">
    <bw:portletcontainer id="root" class="bw-container" px="0" py="0"
          dx="100%" dy="100%" split="V">
      <bw:portletpane id="navi" class="style_viewport_1" px="0" py="0"
          dx="240" dy="100%" radio="1" />
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      <bw:portletcontainer id="c" class="bw-container" px="N" py="0"
           dx="*" dy="100%" split="V">
        <bw:portletpane id="cont" class="style_viewport_1"
                 px="0" py="0"
           dx="*" dy="100%" radio="1" />
        <bw:portletpane id="contr" class="bw_pane"
                 px="N" py="0" dx="200"
           dy="100%" radio="0" />
      </bw:portletcontainer>
    </bw:portletcontainer>
  </bw:portal>

</bw:group>
```

Innerhalb einer Portlet-Datei auf der „äußeren“ Ebene (damit gelten die so definierten Template-Styles für das Portal, auch wenn die Definition innerhalb einer Portlet-Datei angegeben wird !):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:group xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:include src="portalstyles.xml" />

  <bw:portlet id="portlet_id" title="Beispiel" target="cont">
    ...
  </bw:portlet>
</bw:group>
```

Innerhalb der Portlet-Datei auf der „inneren“ Ebene (damit gelten die so definierten Template-Styles lediglich für das Portlet, innerhalb dessen sie inkludiert werden !):

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:group xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:portlet id="portlet_id" title="Beispiel" target="cont">

    <bw:include src="portletstyles.xml" />

    ...
  </bw:portlet>
</bw:group>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Die Verknüpfung zwischen einem Element und einer Template-Style-Definition kann auf zwei unterschiedlichen Wegen geschehen:

- mit Hilfe einer Regeln innerhalb der Style-Definition
- mit Hilfe einer Referenz innerhalb der Element-Definition

Eine Regel wird mit Hilfe des „`select`"-Attributs angegeben. Das „`select`"-Attribut erlaubt aktuell die beiden folgenden Spezifikationen:

```
select="tagname='name'"
select="@attribut='wert'"
```

Im ersten Fall wird das Schlüsselwort „`tagname`" verwendet, um einen Elementnamen zu spezifizieren. Beispiel:

```
select="tagname='bw:label'"
```

In diesem Beispiel beziehen sich die innerhalb des Style-Elements definierten Angaben auf alle `<bw:label>` Elemente (je nach Ort der Definition auf *alle* Label-Elemente, alle Labelelemente eines Portals oder alle Label-Elemente eines Portlets). Es ist zu beachten, dass der namespace (also das „`bw:`") mit angegeben werden muss.

Im zweiten Fall kann jedes zulässige Attribut eines Elements sowie ein entsprechender Wert als Regel verwendet werden. Beispiel:

```
select="@id='mylabel'"
```

In diesem Beispiel beziehen sich die innerhalb des Style-Elements definierten Angaben auf dasjenige Element, dessen „`id`"-Attribut den Wert „`mylabel`" hat. Auf diese Weise können alle erlaubten Attribute verwendet werden.

### 9.9.4 <bw:tborder>

Zur Definition von Styles gehört auch das `<bw:tborder>`-Element, welches die Definition eines Rahmens um bestimmte Elemente erlaubt. Diese Definition dient zunächst als Template und wird einem Element wie z.B. einer „`<bw:editor>`" mit Hilfe des „borderref"-Attributs zugewiesen. Beispiel:

```
<bw:tborder id="blueborder">
  <bw:style>flat</bw:style>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:color>#7F9DB9</bw:color>
  <bw:width>1</bw:width>
</bw:tborder>

<bw:editor id="TE_Text" tabindex="2 borderref="blueborder">
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
    eox="125" eoy="145" ecx="-80" ecy="200"/>
</bw:editor>
```

## 9.10 BML-Widget-Beispiel

Im Folgenden ist ein einfaches Beispiel für die Nutzung der unterschiedlichen Widgets dargestellt.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:group xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

<bw:portlet
  id="BWRegistryList"
  title="{t('Registries')}"
  target="navi"
  state="normal"
  mode="normal">

  <bw:list id="BWRegistryList1" tabindex="1" >
    <bw:pos aox="0" aoy="0" acx="1000" acy="1000"
            eox="0" eoy="0" ecx="0" ecy="0" />

      <bw:content>
        <bw:lhead>
          <bw:lc width="45%"><bw:t>Name</bw:t></bw:lc>
          <bw:lc width="27%"><bw:t>Server</bw:t></bw:lc>
          <bw:lc width="28%"><bw:t>Einträge</bw:t></bw:lc>
        </bw:lhead>
        <bw:lbody>
          <bw:ldata element="Registries.Array.*Registry">
            <bw:li id="{.Name}">
              <bw:lc><bw:var expr=".Name"/></bw:lc>
              <bw:lc><bw:var expr=".Location.Name"/></bw:lc>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
                <bw:lc><bw:var expr=".Size"/></bw:lc>
              </bw:li>
            </bw:ldata>
          </bw:lbody>
        </bw:content>
        <bw:events>
        <bw:event method="{concat( 'handleEvent( selectRegistry,' ,
          concat( PID, ',' , '@nid' , ')' ))}" name="selected"/>
        </bw:events>

  </bw:list>
</bw:portlet>
</bw:group>
```

# 9.11 *B-Wise* Layouter

### 9.11.1 AbsoluteLayout

Der *B-Wise* Layouter stellt die Basis für eine äußerst flexible Positionierung sowie Adaptation der Widgets an die Umgebung (Parent) an. Mit Hilfe des Layouters werden Elemente (Fenster, Widgets, Labels etc.) auf dem „Untergrund“ (z.B. Panel oder Compound) platziert. Neben der statischen (oder initialen) Position kann festgelegt werden, wie sich das Element bei Größenänderungen des Parents verhalten soll (im Folgenden als „Framing“ bezeichnet).

Der *B-Wise*-Layouter steht für alle unterstützten Clients zur Verfügung, so dass in allen Fällen (Smartphones, Tablets, Browser oder SmartClient) dieselbe Flexibilität unterstützt wird.

Die folgende Abbildung veranschaulicht schematisch die Arbeitsweise des *B-Wise*-Layouters. Dabei wird angenommen, dass das <bw:pos>-Element wie folgt aussieht:

```
<bw:pos aox="0" aoy="0" acx="500" acy="1000"
        eox="20" eoy="20" ecx="16" ecy="-60" />
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

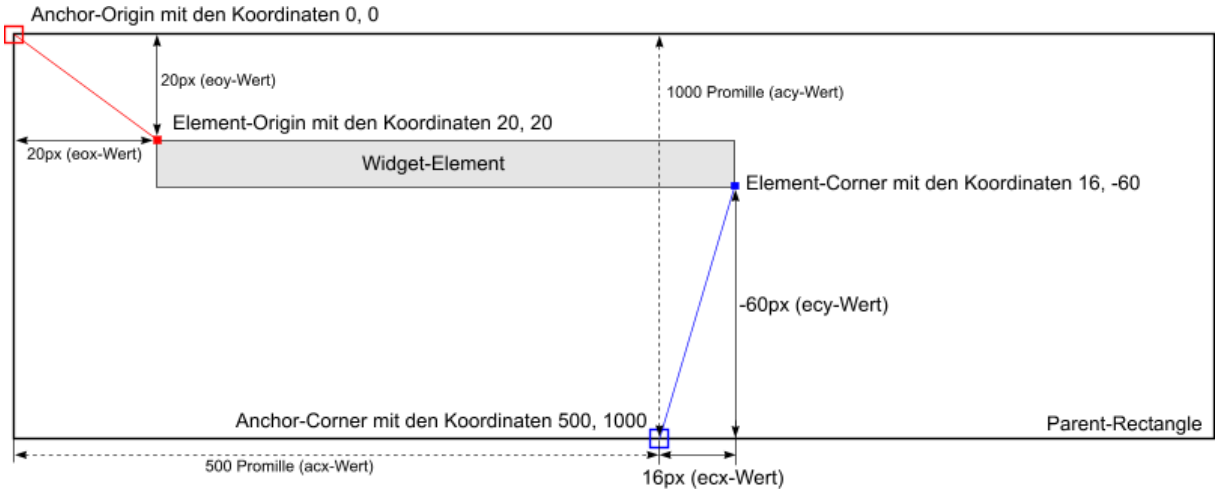



**Abb. 49: *B-Wise* Layouter-Funktionsweise**

Der *B-Wise* Layouter verwendet zwei Bezugspunkte, die im folgenden Anchors (Anker) genannt werden. Einen Anchor-Origin (links oben, in der obigen Abbildung als rotes nicht ausgefülltes Rechteck dargestellt) und einen Anchor-Corner (rechts unten, in der Abbildung als blaues nicht ausgefülltes Rechteck dargestellt). Die Anchors werden immer in Bezug auf den Parent-Origin (linke obere Ecke des Parents) mit Hilfe von x- und y-Angaben in Promille festgelegt. 0, 0 entspricht dabei der linken oberen Ecke des Parents, 1000, 1000 der rechten unteren Ecke des Parents. In der Abbildung 47 sind daher die Koordinaten des Anchor-Corner-Punktes (blau) 500, 1000, d.h. 50% bzw. 500 Promille auf der X-Achse und 100% bzw. 1000 Promille auf der Y-Achse. Die Koordinaten der Anchor-Punkte sind immer positiv, der Ursprung ist immer links oben.

Das Element selbst wird mit Hilfe absoluter Koordinaten in Bezug auf die Anchor-Punkte positioniert. Dabei sind alle Werte nach rechts bzw. unten positiv und alle Werte nach links bzw. oben negativ. Die Positionsangaben erfolgen in Pixel oder Sysfont-Units (in Abhängigkeit des Attributs „unit“ des „<bw:pos>“-Elements, das die Werte „pixel“ bzw. „sysfont“ annehmen kann, wobei „pixel“ der Default-Wert ist). In der obigen Abbildung betragen die Element-Origin-Koordinaten (also die Koordinaten der linken oberen Ecke des Widget-Elements) daher 20, 20 und die Koordinaten der rechten unteren Ecke 16, -60.

Die Anchor-Punkte definieren primär das Verhalten des positionierten Elements bei Größenänderungen des Parents. Im obigen Beispiel wird das Element bei einer Breiten-Änderung des Parents um 100 Pixel daher um 50px geändert (also 50%). Bei einer Höhen-Änderung des Parents um 100 Pixel wird das Element um 100 Pixel (also 100%) geändert.

Um ein Element fix und unveränderlich zu positionieren sind beide Anchor-Punkte mit den Koordinaten 0, 0 zu versehen.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Mit Hilfe der Attribute „hmin“ (für „minimal height“) und „wmin“ (für „minimal width“) kann die minimale Breite (wmin) bzw. minimale Höhe (hmin) eines Elements festgelegt werden.

### 9.11.2 Flow-Layout (auch relatives Layout genannt)

Das bisher beschriebene Verhalten des Layouters bezog sich auf die absolute Positionierung eines Elements, also die explizite Angabe der Position sowie der Größe (implizit durch Bestimmung der beiden Ecken „links/oben“ und „rechts/unten“). Diese Art der Positionierung ist die häufigere bei interaktiven Dialogen, insbesondere, wenn eine gewisse Komplexität des Dialogs erreicht worden ist.

In bestimmten Fällen ist jedoch auch die relative Positionierung (manchmal auch „flowlayout“ genannt) sinnvoll. Dabei wird ein Element nicht abolut innerhalb der Grenzen seines Parents positioniert, sondern relativ zur Position seines Vorgängers. BML kennt die vertikale und die horizontale, relative Positionierung. Beide werden durch Verwendung des Attributs „rel“ des <bw:pos>-Elements definiert, wobei für die vertikale der Wert „yflow“ und für die horizontale der Wert „xflow“ für das Attribut „rel“ angegeben werden muss.

Das folgende Beispiel möge den Sachverhalt verdeutlichen:

```
<bw:label id="LA_Info" align="TopLeft">
  <bw:pos aox="0" aoy="0" acx="1000" acy="0"
    eox="20" eoy="33" ecx="-34" ecy="90" />
  <bw:content><bw:t>Eine Suche VUs ist derzeit nur über die Eingabe
    des Firmennamens möglich. Möchten Sie alle VUs sehen, so lassen
    Sie das Feld Firmenname frei.</bw:t>
  </bw:content>
</bw:label>

<bw:entry id="TF_Name" tabindex="1" datasrc="Nachname">
  <bw:pos rel="yflow" aox="0" aoy="0" acx="1000" acy="0"
    eox="100" eoy="5" ecx="-20" ecy="27" />
  <bw:content><bw:t>Firmenname</bw:t></bw:content>
</bw:entry>
```

Das Eingabefeld “TF_Name” wird vertikal relativ zu seinem Vorgänger-Element „LA_Info“ positioniert. Erreicht wird dies durch Angabe des Attributs rel=“yflow“ innerhalb des <bw:pos>-Elements. Dabei wechselt jedoch die Semantik der beiden Attribute „eoy“ und „ecy“. „eoy“ bestimmt den Abstand zur Unterkante des vorherigen Elements (im Beispiel also 5 Pixel unterhalb der Unterkante des Label-Elements), „ecy“ bestimmt die Höhe des Elements bezogen auf „eoy“ (also 22 Pixel, berechnet aus 27 Pixel minus 5 Pixel).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Die relative Positionierung macht insbesondere Sinn, wenn z.B. der Text eines Labels nicht statisch, sondern dynamisch definiert ist, also sich erst zur Laufzeit der Anwendung und dann auch stark unterschiedlich ergibt. In diesem Fall wird jedoch auch eine Möglichkeit benötigt, die Höhe des Labels dynamisch zu definieren. Das <bw:pos>-Elements (im nachfolgenden Beispiel eines Labels) Label verfügt hierzu über das Attribut „asmode“, welches die drei Werte „none“ (Default), „height“ oder „width“ annehmen kann. Dabei bedeutet „heigth“, dass sich die Höhe des Labels aus dem Inhalt (beim Label also Text) berechnet. Analog bestimmt die Angabe asmode=“width“, dass sich die Breite aus dem Inhalt berechnet. Der jeweils andere Wert muss konstant sein, damit eine Berechnung möglich wird.

Das nachfolgende Beispiel möge den Sachverhalt verdeutlichen:

```
<bw:label id="LA_Info" align="TopLeft">
  <bw:pos asmode="height" aox="0" aoy="0" acx="1000" acy="0"
    eox="20" eoy="33" ecx="-20" ecy="5" />
  <bw:content><bw:var expr="InfoText" /></bw:content>
</bw:label>

<bw:entry id="TF_Name" datasrc="Nachname" tabindex="1">
  <bw:pos rel="yflow" aox="0" aoy="0" acx="1000" acy="0"
    eox="100" eoy="5" ecx="-20" ecy="27" />
  <bw:content><bw:t>Firmenname</bw:t></bw:content>
</bw:entry>
```

Bei der Angabe des Attributs asmode=“height“ ändert sich die Semantik von „ecy”. Das Attribut definiert in diesem Fall den Abstand zwischen dem Inhalt und der Unterkante des Elements (also quasi ein „padding-bottom“ in der CSS-Sprache).

Bei der Angabe des Attributs asmode=“width“ ändert sich die Semantik von „ecx”. Das Attribut definiert in diesem Fall den Abstand zwischen dem Inhalt und der rechten Kante des Elements (also quasi ein „padding-right“ in der CSS-Sprache).

Auch eine gemischte Verwendung von absoluter und relativer Positionierung ist mit Hilfe der o.g. Attribute in Zusammenspiel mit Compounds möglich. So kann z.B. nach einem in der Höhe variablen Label ein relativ positioniertes Compound definiert werden, dessen Inhalte wiederum absolut positioniert sind.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.12 PopUp-Dialoge (modal bzw. nicht-modal)

*B-Wise*-Prozesse verwenden als GUI-Elemente grundsätzlich Portlets, die innerhalb eines Portals platziert und angezeigt werden. Dies erfolgt unabhängig davon, ob es sich dabei um ein Hauptfenster, ein nicht-modales Zusatzfenster oder einen modalen Dialog handelt.

Normalerweise werden Portlets innerhalb des Hauptfensters zur Anzeige gebracht. Vielfach ist es jedoch wünschenswert oder gar erforderlich, Portlets in separaten Fenstern anzuzeigen. Dies sowohl in modaler wie auch nicht-modaler Form. Ob ein Portlet innerhalb des Hauptfensters, eines nicht-modalen Zusatzfensters oder innerhalb eines modalen Dialogs dargestellt werden soll, erfolgt innerhalb des *B-Wise* Prozesses. Prozess bestimmt somit sowohl das Ziel (Fenster) als auch (im Falle eines Zusatzfensters) dessen Modalität.

Sowohl nicht-modales Zusatzfenster als auch modaler Dialog funktionieren exakt wie das Hauptfenster. Der Prozess wählt das gewünschte Portal (besser: die gewünschte Portal-Definition) innerhalb dessen die Portlets platziert werden. Was lediglich noch fehlt, ist die Definition der gewünschten Fenster- bzw. Dialog-Eigenschaften wie z.B. Größe, Position, Menü etc.

Dies wird mit Hilfe einer XML-Datei definiert, die beim vom Prozess (analog dem Portal) gewählt werden kann. Zur Veranschaulichung zeigt das nachfolgende Code-Fragment einen Ausschnitt aus einem Prozess, der einen modalen Dialog anzeigt (die relevanten Methoden sind dabei fett hervorgehoben):

```
class PopUpDlgTest extends GUIProcess {

  public PopUpDlgTest() {
    super();
  }

  public String getName() {
    return "Beispiel-PopUp-Dialog";
  }

  public String getPreferredWindow() {
    return "Test_Dialog.wnd";
  }

  public String getPreferredPortal() {
    return "System_SinglePane.por";
  }
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  public PortletDescrs definePortlets() {

    PortletDescrs portlets = super.definePortlets();
    portlets.add( "Test_Dialog.plt");

    return portlets;
  }

  public void initialize() {
    super.initialize();
  }

  public void start() {
    super.start();
  }
}
```

Der folgende Code enthält ein Beispiel dieser Fenster-Definitionsdatei.

```
<window id="Test_Dialog" title="Titel des Fensters">
  <decoration sizeable="true" />
  <geometry>
    <centered width="638" height="405"/>
  </geometry>
</window>
```

Diese Definitionsdatei beinhaltet insbesondere die Maße des Fensters sowie dessen Position (alternativ die Angabe, ob das Fenster zentriert zur Anwendung oder zum Bildschirminhalt dargestellt werden sollen). Zusätzlich können Angaben zum Styling des Fensters (z.B. Sizeborder) gemacht werden.

Zusätzlich kann für jedes zusätzliche Fenster (modal wie nicht-modal) definiert werden, ob es ein PullDown-Menü, eine Toolbar sowie eine Statuszeile enthalten soll. Im Falle des Menüs und der Toolbar kann auch der initiale Inhalt definiert werden. Im Fall der Statuszeile kann lediglich festgelegt werden, ob die Statutszeile enthalten sein soll oder nicht; der Grund hier liegt darin, dass eine uniforme Behandlung der Statuszeile sowohl für das Hauptfenster als auch alle Zusatzfenster erfolgen soll und dass die Statuszeile des Hauptfensters eine fixe Struktur hat, die in den Zusatzfenstern lediglich wiederholt werden kann.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.13 Scripting (Client-seitiges)

*B-Wise XA* BML erlaubt das Einbetten und Verwenden von Scripten (Code) innerhalb von BML-Dialogseiten (Scripte in der unten beschriebenen Form sind aktuell nicht in Dokumenten möglich), die innerhalb des Clients ausgeführt werden. BML-Designer können Scripte nutzen, um eine Reihe von Aufgaben innerhalb von Dialogen zu erfüllen. Hierzu gehören u.a.

- Sichtbar- bzw. Unsichtbarmachen von Elementen oder Elementgruppen
- Programm-gesteuertes Anzeigen der BubbleHelp
- Anzeigen von Inhalten in unterschiedlichen Styles (z.B. Farben) in Abhängigkeit vom Wertebereich (z.B. alle negativen Werte in rot anzeigen)
- Auf- und Zuklappen von Dialoganteilen durch Auswahl von Elementen (z.B. das Aufklappen eines Zusatzdialogs, wenn der Benutzer in einer Choice den Wert „Ja“ auswählt)

Scripte sollten ausschließlich der Implementierung dynamischer Verhaltensweisen der Benutzer-Oberfläche dienen und nicht zur Implementierung von Logiken genutzt werden (auch wenn dies technisch möglich ist, widerspricht es der *B-Wise XA* Architektur bzw. klaren Trennung von Benutzer-Oberfläche und Logik). Die Client-seitigen Scripte laufen im Kontext des Clients ab (wie der Name bereits impliziert), d.h. haben lediglich Zugriff auf die vom Server zum Client übertragenen Daten (im Ggs. zu Scripten, die während der Transformationen innerhalb des Servers ablaufen, die ihrerseits Zugriff auf das gesamte Document Object Modell - DOM - im aktuellen Kontext des Servers haben).

Als Sprache zur Implementierung der Scripte dient JavaScript. Soll ein Dialog mit eingebetteten Script ausschließlich innerhalb des SmartClients (oder des EmbeddedClients, der in dieser Hinsicht mit dem SmartClient identisch ist) verwendet werden, kann auf die Spracherweiterungen zurückgegriffen werden, die in Microsoft JScript.Net implementiert sind.

JScript bzw. JScript.Net ist die Implementierung der JavaScript-Skriptsprache durch Microsoft®. Es handelt sich um eine vollständige Implementierung mit einigen zusätzlichen Erweiterungen zur Unterstützung der Fähigkeiten des Microsoft Internet Explorers bzw. zur Unterstützung und Einbettung der Scripte in die .net-Umgebung.

JScript.Net ist in *B-Wise XA* aktuell (Stand *B-Wise XA*, 4.1) in der Version 8.0 implementiert. Es handelt sich bei JScript.Net (im Ggs. zu JavaScript) um eine echte objektorientierte Sprache, deren "Skriptcharakter" dennoch erhalten bleibt. JScript 8.0 gewährleistet vollständige Abwärtskompatibilität mit früheren Versionen von JScript und somit auch JavaScript (ECMAScript 262), enthält allerdings wichtige neue Features und bietet Zugriff sowohl auf die Common Language Runtime als auch auf das .net Framework. Eine einführende deut-

© Copyright BISS GmbH, 2001-2012 All Rights reserved

sche Beschreibung findet sich im Internet unter der URL http://msdn2.microsoft.com/de-de/library/72bd815a(vs.80).aspx.

Durch die Kombination der Features früherer Versionen von JScript bzw. JavaScript mit den effizientesten Features klassenbasierter Sprachen schafft JScript 8.0 optimale Synergien. Die Verbesserungen in JScript 8.0, das in Verbindung mit ECMAScript Edition 4 entwickelt wird, umfassen echten kompilierten Code, typisierte und typenlose Variablen, spätes und frühes Binden, Klassen (mit Vererbung, Überladen von Funktionen, Eigenschaftenaccessoren usw.), Pakete, sprachübergreifende Unterstützung und vollständigen Zugriff auf .net Framework.

**Hinweis:** Soll der entsprechende Dialog mit eingebettetem Script-Code sowohl innerhalb des SmartClients (bzw. EmbeddedClients) als auch innerhalb eines Nicht-Internet Explorer BrowserClients (also z.B. innerhalb von FireFox) ablauffähig sein, können die oben definierten Erweiterungen (z.B. typisierte Variablen oder Klassen) *nicht* verwendet werden. Hier ist die ausschließliche Nutzung derjenigen JavaScript-Sprachelemente und -Funktionen zwingend, die in ECMA-Script 262 definiert sind (diese sind allgemein unter JavaScript bekannt, genauer unter JavaScript 1.5).

Scripte haben Zugriff auf innerhalb des Clients existierende (GUI-) Elemente, z.B. Controls, wie Eingabefelder oder Buttons sowie deren Inhalte (z.B. den Inhalt eines Eingabefeldes). Scripte können zusätzlich das API dieser Elemente verwenden, um z.B. ein Element sichtbar oder unsichtbar zu machen oder die BubbleHelp (die sog. Tooltips) explizit anzuzeigen. Welche API-Funktionen bei jedem Element verfügbar sind, ist bei der Beschreibung eines jeden Elements dokumentiert.

Eine detaillierte Beschreibung des `<bw:cjscript>`-Elements sowie dessen Verwendung enthält der folgende Abschnitt 9.13.1.

**Hinweis:** Das `<bw:cjscript>`-Element sollte nicht mit dem `<bw:script>`-Element verwechselt werden, das ebenfalls in JavaScript implementierte Scripte definiert, die jedoch im Kontext *des Servers* ausgewertet bzw. ausgeführt werden ! Im Kontext des Servers ausgeführte Scripte stehen sowohl Dialogen als auch Dokumenten zur Verfügung !

Innerhalb des SmartClient und des EmbeddedClients werden Client-seitige Scripts compiliert. Dieser Vorgang kann bei langen und komplexen Scripten durchaus einige Zeit in Anspruch nehmen, so dass komplexe Scripte vermieden werden sollten. Die Scripte werden allerdings erst unmittelbar vor der Ausführung compiliert, so dass es nicht schädlich ist, viele Scripte in einem Dialog zu haben, die nicht alle verwendet werden. Dies kann eintreten, wenn alle Scripte einer Anwendung in einer separaten Datei konzentriert werden, die anschließend in die einzelnen Dialoge inkludiert wird.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.13.1 Hinweise zur Verwendung von Scripten

Wenn Sie einen Dialog in BML definieren, können Sie Scripte z.B. zur Definition dynamischer Vorgänge innerhalb des Dialogs oder zur Verarbeitung von Daten unmittelbar vor deren Anzeige verwenden. So können Sie z.B. ein Berechnungsergebnis in Abhängigkeit von dessen Wert in schwarz (Wert >= 0) oder rot (Wert < 0) darstellen.

Scripte können direkt innerhalb der BML-Definitionsdatei eingebettet oder in einer externen Datei definiert sein, die anschließend von der BML-Dialogdatei inkludiert wird. Letzteres empfiehlt sich für Scripte, die querschnittlich von mehreren Dialogen verwenden werden sollen.

`<bw:cjscript>`-Blöcke dürfen ausnahmslos Funktionsdefinitionen enthalten. Die Deklaration oder Definition von Variablen oder das Verwenden von Statements außerhalb von Funktionen ist nicht erlaubt und führt zu einem Fehler, der alle Funktionen innerhalb des Kontextes invalidiert ! Ein einziger Fehler an einer beliebigen Stelle führt somit zum Ungültigwerden alle Scripte innerhalb des aktuellen Dialogs.

Der Inhalt der `<bw:cjscript>`-Blöcke sollte grundsätzlich innerhalb einer CDATA-Section stehen, um Probleme mit Sonderzeichen wie z.B. dem Kleiner-Symbol („<“) während des Parsens der Datei zu vermeiden. Streng genommen ist das Einbetten in eine CDATA-Section nur dann erforderlich, wenn Sonderzeichen verwendet werden, die zu Parse-Konflikten des XML-Parsers führen; allerdings ist es guter Stil, dies grundsätzlich zu tun.

Ein wesentliches Merkmal von Client-seitigen Scripten ist der Zugriff auf die im aktuellen Dialog-Kontext verfügbaren Elemente. Scripte können auf fast alle Elemente, die sich auf dem aktuellen Dialog befinden, zugreifen. Hierzu muss das Element eine eindeutige ID haben. Sie verwenden anschließend die Methode „`getElementById()`“ des „`Doc`“-Objektes, um eine Referenz auf das gewünschte Element zu erhalten. Anschließend können Sie sowohl Eigenschaften des Elements (z.B. „`visibility`“) abfragen als auch setzen. Zusätzlich können fast alle Methoden des jeweiligen Dialogelements aufgerufen werden (z.B. „`showBubbleHelp()`“).

Um eine Eigenschaft eines Elements abzufragen, verwenden Sie die Methode „`getProperty()`“ im Zusammenhang mit der Referenz auf das Element. Der folgende Beispielcode fragt ein `<bw:compound>`-Element mit der `id="CP_1"` auf dessen Sichtbarkeit ab:

```
<bw:cjscript>
  function getState() {
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    var el = Doc.getElementById( "CP_1");
    var vis = el.getProperty( "visibility");
  }
</bw:cjscript>
```

Um eine Eigenschaft eines Elements zu setzen, verwenden Sie die Methode „`setProperty()`“ des Elements. Der folgende Beispielcode setzt die Eigenschaft „`visibility`“ des Compound-Elements auf „`false`“, d.h. macht das Compound-Element (und damit auch alle seine Kindelemente) unsichtbar:

```
<bw:cjscript>
  function setState() {
    var el = Doc.getElementById( "CP_1");
    el.setProperty( "visibility", false);
  }
</bw:cjscript>
```

Selbstverständlich sind auch kombinierte oder weitaus komplexere Funktionen möglich; das folgende Script schaltet ein Compound-Element zwischen den Zuständen „sichtbar“ und „unsichtbar“ hin und her:

```
<bw:cjscript>
  function toggleState() {
    var el = Doc.getElementById( "CP_1");
    el.setProperty( "visibility", !el.getProperty( "visibility");
  }
</bw:cjscript>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

# 9.13.2 <bw:cjscript>

### Kurzbeschreibung

`<bw:cjscript>` dient der Einbettung von in JavaScript bzw. JScript.Net geschriebenen Scripten in eine Dialogseite.

Die so implementierten Scripte werden normalerweise mit Hilfe von Events aufgerufen. Hierzu erhält das `<bw:event>`-Element ein zusätzliches Attribut, dass die Verwendung der im Attribut `method` definierten Funktion als Script-Funktion definiert (siehe hierzu auch den Abschnitt 9.3.2.4, Event-Handling). Ein Beispiel hierzu findet sich weiter unten. Das Script selbst bzw. dessen einzelne Funktionen sind innerhalb des `<bw:cjscript>`-Elements implementiert.

### Syntax

```
<bw:cjscript>
  ... Script-Definition (Implementierung)
</bw:cjscript>
```

### Parameter

Das `<bw:cjscript>`-Element hat keine Attribute.

### API-Funktionen

Das `<bw:cjscript>`-Element verfügt über keine API-Funktionen.

### Beispielverwendung

```
<bw:cjscript>
  <![CDATA[
  function showTip( val) {
    var el = Doc.getElementById( val);
    el.showBubbleHelp( 2);
  }
  ]]>
</bw:cjscript>
```

Im dargestellten Beispiel wurde bewusst auf eine sinnvolle Fehlerbehandlung verzichtet, um die wesentliche Elemente eines einfachen Scripts herauszustellen.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Zunächst wird das Script in eine `<![CDATA[`-Section eingebettet, da ansonsten innerhalb des Scripts verwendete Sonderzeichen (z.B. das „<“-Zeichen für den logischen Vergleich) vom XML-Parser als Fehler angekreidet werden würden.

Das Script selbst verwendet das `Doc`-Objekt des Clients (also das Document-Objekt, das dem Dialog entspricht - nicht zu verwechseln mit einem Papier - bzw. PDF-Dokument !) bzw. dessen Methode „`getElementById`“, um zunächst eine Referenz auf das Element mit der id „`val`“ zu bekommen. Diese wird beim Aufruf (siehe unten) als Argument an die Funktion geliefert. Anschließend wird die Methode „`showBubbleHelp`“ des Elements aufgerufen; der als Argument an diese Methode übergebene Wert „2“ bestimmt die Anzeigedauer der BubbleHelp in Sekunden.

Der Aufruf erfolgt aus einem Event, im vorliegenden Beispiel aus einem Eventhandler eines `<bw:picturebox>`-Elements, welches als Fragezeichen hinter dem Element angezeigt wird (siehe auch den nachfolgenden Ausschnitt eines Screenshots).

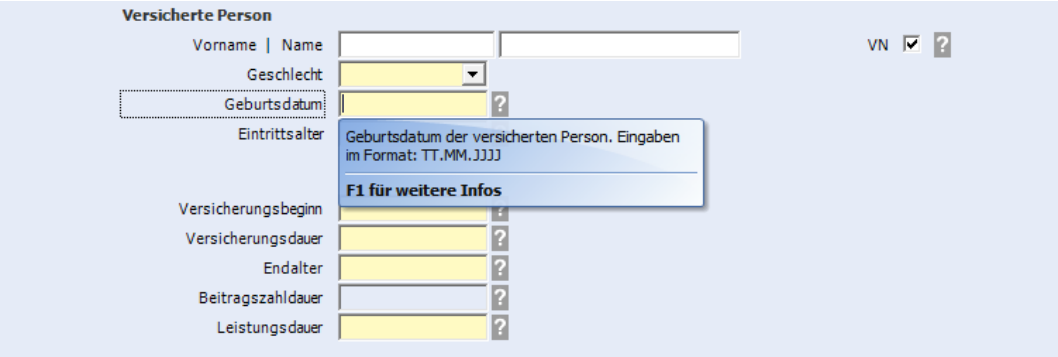



**Abb. 50: Screenshot des Beispiel-Dialogs (Ausschnitt)**

Der zugehörige BML-Code sieht wie folgt aus:

```
<bw:entry id="TF_T1" tabindex="1">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
    eox="165" eoy="10" ecx="420" ecy="31" />
  <bw:tooltip>
    <bw:ttb><bw:t>Das ist der BubbleHelp-Text</bw:t></bw:ttb>
  </bw:tooltip>
</bw:entry>
<bw:picturebox id="PB_T1">
  <bw:pos aox="0" aoy="0" acx="0" acy="0"
    eox="425" eoy="10" ecx="437" ecy="31" />
  <bw:img src="images/dlghelp.gif" />
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  <bw:events>
    <bw:event name="clicked" type="cjscript" method="showTip( TF_T1)"/>
  </bw:events>
</bw:picturebox>
```

Die wesentliche Erweiterung gegenüber einem „normalen“ Event-Handler ist das Attribut „`type`“, dessen (aktuell fixer) Wert „`cjscript`“ signalisiert, dass es sich bei der im Attribut „method“ definierten Methode nicht um eine innerhalb des *B-Wise XA* Servers implementierte sondern um eine Script-Methode handelt.

**Hinweis:** Aktuell können ausschließlich Methodenaufrufe innerhalb des Eventhandlers verwendet werden. Es ist nicht möglich den Script-Code direkt in das „method“-Attribut einzutragen !

Seiten-spezifische Scripte werden normalerweise innerhalb der Dialogseite mit Hilfe eines eingebetteten `<bw:cjscript>`-Elements definiert. Seiten-übergreifende Scripte werden normalerweise innerhalb einer eigenen Datei definiert, die anschließend mit Hilfe des `<bw:include>`-Elements in eine Dialogseite inkludiert wird.

Der folgende Abschnitt beschreibt die verfügbaren Objekte (z.B. das oben verwendete „Doc“-Objekt) sowie deren Methoden (z.B. die oben verwendete Methode „getElementById“).

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.13.2.1 Objekte und Methoden

Dieser Abschnitt beschreibt die innerhalb der Scripte verfügbaren Objekte und deren Methoden. Zusätzlich zu den hier beschriebenen Objekten und deren Methoden stehen alle Controls als Objekte zur Verfügung. Die darin verfügbaren Eigenschaften (Properties) bzw. aufrufbaren Methoden sind beim jeweiligen Control beschrieben.

> **Hinweis:** Die sog. „Deco"-Elemente <bw:line>, <bw:rect>, <bw:circle> und <bw:img> sind keine Controls, verfügen daher werden über scriptbare Eigenschaften nicht Methoden !

### Global

Eine Methode kann innerhalb eines Skripts auch ohne in passendes Objekt (also „global") verwendet werden.

#### alert

Die alert-Methode zeigt einen beliebigen String innerhalb einer MessageBox und unterbricht die Ausführung des Skripts, bis die MessageBox vom Anwender geschlossen worden ist.

**Syntax**

```
alert( msg);
```

Das Argument „msg" kann grundsätzlich ein beliebig langer String sein, der jedoch je nach Plattform ab einer bestimmten Länge abgeschnitten wird.

### Doc

Das `Doc`-Objekt repräsentiert den eigentlichen Dialog, d.h. das innerhalb des Clients (SmartClient, EmbeddedClient oder BrowserClient) dafür aufgebaute Document Object Model (DOM).

Wenn der Client eine Dialogbeschreibung in BML empfängt, erstellt er hieraus eine interne Repräsentation in Form eines DOMs. Die einzelnen Elemente des DOMs stellen die Elemente des Dialogs dar (z.B. Eingabefeld oder Button).

Das `Doc`-Objekt stellt aktuell zwei Methoden für den grundsätzlichen Zugriff auf die Elemente dieses DOMs bzw. das Neulayouten des Dialog zur Verfügung.

#### Doc.getElementById

Diese Methode ermöglicht den gezielten Zugriff auf ein Element des DOMs (also z.B. auf eine Eingabefeld innerhalb des Dialogs) mit Hilfe der Element-ID.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Syntax**

```
Object = Doc.getElementbyId( id);
```

Das String-Argument `id` stellt die innerhalb des Dialogs eindeutige ID des gewünschten Elements dar. Die Methode liefert eine Referenz auf das zugehörige Element.

**Beispiel:**

```
var el = Doc.getElementById( "TF_Nachname");
```

Im obigen Beispiel wird der Variablen „`el`“ eine Referenz auf das innerhalb des Dialogs vorkommende Element mit der ID „TF_Nachname“ zugewiesen. Hat sich der Dialog-Designer an die Namenskonventionen gehalten, handelt es sich hier um ein Eingabefeld (TF = Textfield). Mit Hilfe dieser Referenz können nun Methoden des Eingabefeldes (vom Typ `<bw:entry>`) aufgerufen werden, z.B. die Methode „`setVisible`“. Die innerhalb eines `<bw:entry>` verfügbaren Methoden sind bei der Dokumentation dieses Controls weiter oben beschrieben.

**Doc.doLayout()**

Diese Methode ermöglicht ein explizites Relayouten der aktuellen Dialogseite. Der Client wird veranlasst, den initial nach Empfang und Decodierung der BML-Seiten durchgeführten Layout-Prozess nochmals auszuführen.

**Syntax**

```
Doc.doLayout();
```

Die Methode hat weder Argumente noch liefert sie einen return-Wert zurück.

**Beispiel:**

```
Doc.doLayout();
```

Diese Methode kann verwendet werden, wenn Scripte sehr viele Modifikationen an vielen Dialogelementen durchführen und daher ein vollständig neues Layout schneller ist als eine Vielzahl von Relayouts der einzelnen Elemente.

**Doc.processCommand()**

Diese Methode ermöglicht die Ausführung einer Methode auf dem Server. An die Methode können auch Argumente übergeben werden.

**Syntax**

```
Doc.processCommand( methodname, args);
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Das erste Argument definiert den Namen der auszuführenden Methode, das zweite Argument definiert die Argumente, die an die Methode übergeben werden sollen.

**Beispiel:**

```
Doc.processCommand( "doThis", false);
```

**Doc.debug()**

Diese Methode ermöglicht es, beliebige Informationen auf der Debug-Konsole des Clients auszugeben und dient damit – wie der Name bereits suggeriert – primär Debugzwecken.

**Syntax**

```
Doc.debug( arg);
```

Das Argument definiert einen beliebigen String.

**Beispiel:**

```
void Doc.debug( "Bis hierher fehlerfrei gekommen");
```

### Elemente

Die nachfolgenden Methoden können zusammen mit einem beliebigen BML-Element verwendet werden, mit Ausnahme der Deco-Elemente (line, rect, circle, img, groupbox). Um dies anzudeuten, wird als Objektname in den folgenden Beispielen „Elem“ verwendet.

**Elem.getProperty**

Die Methode getProperty gibt den Wert der als Argument übergebenen Eigenschaft des Elements zurück.

**Syntax**

```
Object = Elem.getProperty( propname);
```

Das Argument definiert den Namen der Eigenschaft, die abgefragt werden soll. Die Methode gibt den Wert der Eigenschaft zurück, der Typ der Rückgabe ist je nach Eigenschaft unterschiedlich.

**Beispiel**

```
var tf = Doc.getElementById( "TF_Name");
var vis = tf.getProperty( "Visible");
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Das Beispiel weist der boolschen Variablen „vis“ den Wert der Eigenschaft „Visible“ des Eingabefelds mit der ID = „TF_Name“ zu.

**Elem.setProperty**
Die Methode setProperty setzt den Wert der als Argument übergebenen Eigenschaft des Elements auf den als zweites Argument übergebenen Wert.

**Syntax**
```
void Elem.setProperty( propname, propvalue);
```

Das erste Argument definiert den Namen der Eigenschaft, die gesetzt werden soll. Das zweite Argument definiert den Wert. Die Methode gibt void zurück.

**Beispiel**
```
var tf = Doc.getElementById( "TF_Name");
tf.setProperty( "Visible", false);
```

Das Beispiel macht das Eingabefeld mit der ID = „TF_Name“ unsichtbar.

**Elem.addEventHandler**
Die Methode fügt einem definierten Event des Element eine Methode zu.

**Syntax**
```
void Elem.addEventHandler( eventname, method);
```

Das erste Argument definiert den Namen des Events, dem die Methode „method“ (zweites Argument) hinzugefügt wird.

**Beispiel**
```
var tf = Doc.getElementById( "TF_Name");
tf.addEventHandler( "validate", "doThis");
```

Das Beispiel fügt dem Eventhandler „validate“ des Eingabefeldes „TF_Name“ die Methode „doThis“ hinzu.

**Elem.triggerEvent**
Die Methode triggert explizit ein definiertes Event eines Elements und führt damit alle darin registrierten Methoden aus.

**Syntax**
```
void Elem.triggerEvent( eventname);
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Das Argument definiert den Namen des Events, das getriggert werden soll.

**Beispiel**

```
var tf = Doc.getElementById( "TF_Name");
tf.triggerEvent( "validate");
```

Das Beispiel triggert das Event „validate“ des Eingabefeldes „TF_Name“ und führt damit alle darin registrierten Methode aus.

**Elem.setStyleRef**

Die Methode weist dem Element eine Style-Referenz zu.

**Syntax**

```
void Elem.setStyleRef( refid);
```

Das Argument definiert die ID der Style-Definition. Die Style-Definition selbst wird innerhalb des BML-Codes mit Hilfe des `<bw:tstyle>`-Elements durchgeführt.

**Beispiel**

```
var tf = Doc.getElementById( "TF_Name");
tf.setStyleRef( "bigandbold");
```

Das Beispiel wird dem Eingabefeld „TF_Name“ der Style „bigandbold“ zugewiesen.

**Elem.showBubbleHelp**

Die Methode zeigt die BubbleHelp des Elements für eine definierte Zeit an.

**Syntax**

```
void Elem.showBubbleHelp( time);
```

Das Argument definiert die Anzeigedauer der BubbleHelp in Sekunden.

**Beispiel**

```
var tf = Doc.getElementById( "TF_Name");
tf.showBubbleHelp( 5);
```

Das Beispiel zeigt für das Eingabefeld „TF_Name“ die BubbleHelp für 5 Sekunden an.

**Elem.setFocus**

Die Methode setzt den Focus auf das Element.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Syntax**

```
void Elem.setFocus();
```

Die Methode hat keine Argumente.

**Beispiel**

```
var tf = Doc.getElementById( "TF_Name");
tf.setFocus();
```

Das Beispiel setzt den Focus auf das Eingabefeld „TF_Name“.

**Elem.getPreferredWidth**
Die Methode.

**Syntax**

```
int Elem.getPreferredWidth();
```

Die Methode hat Argumente.

**Beispiel**

```
var tf = Doc.getElementById( "TF_Name");
var w = tf.getPreferredWidth();
```

Das Beispiel gibt die bevorzugte Breite des Eingabefelds „TF_Name“ zurück und weist sie der Variablen „w“ zu.

**Elem.getPreferredHeight**
Die Methode.

**Syntax**

```
int Elem.getPreferredHeight();
```

Die Methode hat Argumente.

**Beispiel**

```
var tf = Doc.getElementById( "TF_Name");
var h = tf.getPreferredHeight();
```

Das Beispiel gibt die bevorzugte Höhe des Eingabefelds „TF_Name“ zurück und weist sie der Variablen „h“ zu.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Elem.getLocation**
Die Methode gibt ein Positionsobjekt mit den Eigenschaften „X“ und „Y“ zurück, die der linken, oberen Ecke des Elements entsprechen, bezogen auf das unmittelbare Parent-Element.

**Syntax**
```
Object Elem.getLocation();
```

Die Methode hat Argumente.

**Beispiel**
```
var tf = Doc.getElementById( "TF_Name");
var x = tf.getLocation().X;
```

Das Beispiel gibt die X-Position des Eingabefelds „TF_Name“ zurück bezogen auf das unmittelbare Parent-Element und weist sie der Variablen „x“ zu.

**Elem.layout**
Die Methode erzwingt ein Relayout des Elements und aller seiner Kind-Elemente. Diese Methode wird vor allem auf ein Portlet oder auf ein Compound angewendet.

**Syntax**
```
void Elem.layout();
```

Die Methode hat Argumente.

**Beispiel**
```
var tf = Doc.getElementById( "TF_Name");
tf.layout();
```

Das Beispiel erzwingt ein Relayout des Eingabefelds „TF_Name“.

**Elem.getMargin**
Die Methode liefert die Abstände zwischen dem Element und einem definierten Parent-Element in Pixel. Die zurückgelieferten Werte bezeichnen die Abstände (Margins) jeweils zwischen dem betreffenden Rand des Elements und des Parents, also z.B. "Top" bezeichnet den Abstand zwischen dem oberen Rand des Elements und dem oberen Rand des Parents, usw.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Syntax**

```
void Elem.getMargin( parentid).Top;
```

Die Methode verwendet als optionales Argument die ID (String) des Parents. Wird kein Wert angegeben wird der unmittelbare Parent verwendet. Die vier möglichen Werte sind: „Top“, „Bottom“, „Left“ und „Right“.

**Beispiel**

```
var m = Elem.getMargin( "TF_Name").Top;
var m = Elem.getMargin().Left;
```

Das Beispiel  erzwingt ein Relayout des Eingabefelds „TF_Name“.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.13.2.2 Objekt und Eigenschaften

Nachfolgend sind die Eigenschaften der Elemente aufgezählt. Dabei werden zunächst die Eigenschaften genannt, die allen Elementen geimeinsam sind, anschließend die Element-spezifischen Eigenschaften.

### Allgemein

In diesem Abschnitt werden diejenigen Eigenschaften aufgezählt, die allen Elementen gemeinsam sind.

#### Ctrl

Die „Ctrl"-Eigenschaft entspricht dem auf der Windows-Plattform zugrundeliegenden Control-Element der .Net-Welt. Über diese Eigenschaft können alle weiteren Eigenschaften, Events und Methoden des Elements erreicht und verwendet werden.

**Beispiel:**

```
var label = Doc.getElementById( "LA_Label");
label.Ctrl.Links.Clear();
label.Ctrl.Text = "Neuer Linktext";
label.Ctrl.Links.Add(
     new LinkLabel.Link( 0, label.Ctrl.Text.Length,
     "cjscript:toggleSize()")
);
```

Das Beispiel weist einem Label-Element einen neuen Hyperlink-Text zusammen mit dessen Skrip-Methode zu.

#### Size

Die Size-Eigenschaft enthält zwei weitere Untereigenschaften: „Width" und „Height", die beide die aktuellen Dimensionen des Elements enthalten. Beide können einzeln abgefragt werden.

**Beispiel:**

```
var label = Doc.getElementById( "LA_Label");
var h = label.getProperty( "Size.Height");
var w = label.getProperty( "Size.Width");
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

**PreferredSize**

Die PreferredSize-Eigenschaft enthält zwei weitere Untereigenschaften: „Width“ und „Height“, die beide die aktuellen Dimensionen des Elements enthalten. Beide können einzeln abgefragt werden.

**Beispiel:**

```
var label = Doc.getElementById( "LA_Label");
var h = label.getProperty( "PreferredSize.Height");
var w = label.getProperty( "PreferredSize.Width");
```

**Text**

Die Text-Eigenschaft enthält den Text-Inhalt des Elements (z.B. den eingegebenen Text bei einem Eingabefeld).

**Beispiel:**

```
var label = Doc.getElementById( "LA_Name");
var txt = label.getProperty( "Text");
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

## 9.13.2.3 Script-Beispiele

Nachfolgend sind einige nützliche Scripting-Beispiele aufgeführt:

Ein externer Browser kann wie folgt aus einem Linklabel heraus geöffnet warden (z.B. zur Anzeige von Hilfeinformationen):

```
<bw:cjscript>
<![CDATA[
  function openPage( url ) {
    System.Diagnostics.Process.Start( url);
  }
  ]]>
</bw:cjscript>
```

Ein lokales Skript innerhalb eines Portlets kann auf die globalen Scripte wie folgt zugreifen:

```
retval = Doc.processCommand( "cjscript://doSomething()");
```

Die Methode (im Beispiel "doSomething") kann Argumente übergeben bekommen, die bei Strings in einfachen Hochkommata (') eingeschlossen sein müssen. Innerhalb eines Strings dürfen allerdings keine runden Klammern () sowie nicht die Zeichenfolge :// vorkommen!

Es gibt eine Skript-Methode, über die der Parent-Dialog eines Elements geschlossen werden kann; dies ist insbesondere bei PopUp-Dialoge sinnvoll, deren Close-Ereignis auf dem Server keine Relevanz besitzt (z.B. Dialoge, die als Nachrichten-Dialoge vom Server initiiert worden sind); Beispiel-Eintrag in einem Button:

```
<bw:event name="validate" type="self" method="closeDialog()" />
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Anhang A

# BML-Diskussion

## Diskussion der BML innerhalb von *B-Wise*

**Bestandteile der HTML-Vorlage**

Ein *B-Wise* HTML-Dokument enthält folgende Bestandteile:

- Statische Anteile (Images, Texte, Fonts, Farben, Grobes Layout = Tabellen / Columns / Paragraphs).
- Referenzen (<bw:a href='#'> ... </bw:a>)
  Diese sollen durch konkrete Commands ersetzt werden: aber nur wenn der Benutzer die Berechtigung dafür hat. Ansonsten sollen sie (und potentiell ihr Inhalt) verschwinden.
- Variablen-Inhalte (‚Herr Dr. Jan Mazac')
  Diese sollen durch Zugriffe auf konkrete Daten (‚Partner.Fullname') ersetzt werden. Schon in diesem Beispiel handelt es sich aber um ein virtuelles Feld, welches erst mit einem Algorithmus aus verschiedenen realen Feldern zusammengesetzt wird (‚Partner.Geschlecht', ‚Partner.Titel', ‚Partner.Vorname', ‚Partner.Nachname').
  Zusätzlich soll ein eventueller Status der Daten (fehlerhaft/noch einzugeben/...) berücksichtigt werden, indem z.B. die Textfarbe entsprechend geändert wird.
  Eine Sonderform sind Mappings von Werten (‚privat' -> ‚(priv)').
- Listen (<bw:table><bw:td>...</bw:td><bw:td>...</bw:td>…</bw:table>)
  In der Vorlage sind verschiedene Beispiele für Listenelemente vorhanden. Dies soll durch eine Iteration über die entsprechende Daten-Liste ersetzt werden. Initial soll eine vorgegebene maximale Anzahl von Listenelementen angezeigt werden, dann soll die volle Liste (oder die nächsten n Elemente?) aufgeklappt / zugeklappt werden können (s. auch Blöcke).
  Die Listenelemente sollen optional alternierende Hintergrundfarben haben.
- Ausblendungen
  Bestimmte Sektionen sind nur beispielhaft vorhanden und sollen ignoriert werden.
- Fallunterscheidungen
  Je nach Dateninhalt sollen z.B. Images oder Texte ausgetauscht werden resp. ganze Blöcke unterdrückt werden. Dies ist in der HTML-Vorlage beispielhaft (‚Programming by Examples') vorgegeben, stößt aber sehr schnell an seine Grenzen.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

- Blöcke
  Solche Blöcke werden als Informationseinheit angesehen und sollen auf Klick ausgeblendet / aufgeklappt werden können. Der Aufklapp-Status soll vom Client verwaltet werden aber auch zum Server übertragen werden können resp. initial berücksichtigt werden.

### Identifikation/Markierung der HTML-Bestandteile

Ausgehend von der HTML-Vorlage sollen jetzt die verschiedenen Bestandteile derart identifiziert/markiert werden, dass die ursprüngliche Vorlage erhalten bleibt und weiterhin mit Werkzeugen (z.B. FrontPage) editiert werden kann.

Bei FrontPage ist es möglich, zusätzliche Elemente/Attribute in den HTML-Quelltext einzufügen. Diese werden gutartig ignoriert (sie bleiben auch beim Speichern erhalten).

- Statische Anteile:
  bleiben unverändert.
- Referenzen / Commands:
  Die ursprüngliche <bw:a href="…" /> wird ersetzt durch
  <bw:a href="{**cmd**('EditPartner', Partner.PaID)}" />
- Variablen:
  <bw:**var** name="Partner.Geboren">…</bw:**var**>
- Virtuelle Felder (= Funktionsaufrufe):
  <bw:**var** name="fullName( Partner.Geschlecht,Partner.Titel,..)">...</bw:**var**>
- Mappings:
  Benutzung:
  <bw:**var** name="**lookup**( Kennungen, Partner.Adressen.Kennung,'val')> ...
  </bw:var>
  wobei der Parameter ‚val' auch als Default weggelassen werden kann.
  Definition:
  <bw:**lutab** name="Kennungen">
  <bw:**te** key="Telefon dienstlich" val="Telefon" val2="dienstlich" />
  <bw:**te** key="Telefon privat" val="Telefon" val2="privat" />
  <bw:**te** key="Fax privat" val="Fax" val2="privat" />
  </bw:**lutab**>
- Listen:
  <bw:**foreach** elem="Adresse" inlist="Partner.Adressen"
  maxn="1">...</bw:**foreach**>
- Ausblendungen:
  <bw:**ig**>...</bw:**ig**>


© Copyright BISS GmbH, 2001-2012 All Rights reserved

- Fallunterscheidungen:
  <bw:**if** var="Partner.Adressen[i].Art" value="Hauptwohnsitz">...</bw:**if**>
  oder
  <bw:**if** var="telefonArt( Partner.Telekom[i].Kennung)" value="Fax"> ...</bw:**if**>
- Blöcke:
  Werden mit Fallunterscheidungen gelöst.

### Identifikation/Markierung mit XSLT

Dies setzt voraus, dass die Daten / Berechtigungen in Form von XML vorliegen.

- Statische Anteile:
  bleiben unverändert, werden mit einer generischen Regel behandelt.
- Referenzen / Commands:

```
<bw:xsl:choose>
  <bw:xsl:when test="/grant/cmd/@name = 'EditPartner'">
    <bw:xsl:element name="a">
      <bw:xsl:attribute name="href">
        <bw:xsl:value-of select="js:cmd( 'EditPartner', number( position()),
string( /PartnerPers/PaID))" />
        …
      </bw:xsl:attribute>
    </bw:xsl:element>
  </bw:xsl:when>
  <bw:xsl:otherwise>
    …
  </bw:xsl:otherwise>
</bw:xsl:choose>
```

- Variablen:

```
<bw:xsl:choose>
  <bw:xsl:when test="/Partner/Geboren[@Status]">
    <bw:xsl:element name="font">
      <bw:xsl:attribute name="color">
        <bw:xsl:value-of select="js:statusColor(/Partner/Gebore/@Status)" />
      </bw:xsl:attribute>
      <bw:xsl:value-of select="/Partner/Geboren" />
    </bw:xsl:element>
  </bw:xsl:when>
  <bw:xsl:otherwise>
    <bw:xsl:value-of select="/Partner/Geboren" />
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    </bw:xsl:otherwise>
  </bw:xsl:choose>
```

- Virtuelle Felder (= Funktionsaufrufe):
  <bw:**xsl:value-of** select="js:fullName( string( /Partner/Geschlecht), string( /Partner/Titel), string( /Partner/Nachname))" />
- Listen:
  <bw:**xsl:for-each** select="/Partner/Adressen/Adresse">…</bw:**xsl:for-each**>

  Die Vorkommnisse der even/odd Styles müssen modifiziert werden:
  <bw:td class="ltablineel">...</bw:td> wird zu

```
<bw:xsl:element name="td">
  <bw:xsl:attribute name="class">
    <bw:xsl:value-of select="js:evenOdd( number( position()),'ltablineel','ltablineol')" />
  </bw:xsl:attribute>
  ...
</bw:xsl:element>
```

  Die Aufklapp-Problematik (n="1") ist noch nicht berücksichtigt.
- Ausblendungen:
  Entweder die entsprechenden Teile werden gelöscht oder als Kommentar beibehalten <bw:**xsl:comment**>...</bw:**xsl:comment**>
- Fallunterscheidungen:
  <bw:**xsl:if** test="/Partner/Adressen/Adresse/Art ='Hauptwohnsitz'">...</bw:**xsl:if**>
  oder
  <bw:**xsl:if** test="js:telefonArt( string(/Partner/Telekom/Telefon/Kennung) ='Fax'">...</bw:**xsl:if**>
- Blöcke:
  <bw:**xsl:if** test="js:group('Stammdaten'>…</bw:**xsl:if**>
  Der Inhalt der Aufklapp-<bw:a href=".."> muss ersetzt werden durch
  <bw:a href="javascript:closeGroup(‚Stammdaten')">


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Anhang B

# BML-DTD

```
<bw:!--
  B-Wise Meta-Language DTD
  Namespace = http://www.biss-net.com/2001/bwise/BML
  (C) Copyright BISS GmbH, 2001, All rights reserved
  $Version: 0.51
  $date: 21.02.2002
  $Author: Dr. Richard Seifert
-->

<bw:!ENTITY %   ElemName      "CDATA">
<bw:!ENTITY %   ListName      "CDATA">
<bw:!ENTITY %   Count         "CDATA">
<bw:!ENTITY %   Expression    "CDATA">
<bw:!ENTITY %   VarName       "CDATA">
<bw:!ENTITY %   FileName      "CDATA">
<bw:!ENTITY %   ClsName       "CDATA">
<bw:!ENTITY %   BlockName     "CDATA">

<bw:!--=========== Loop Element =============-->
<bw:!ELEMENT foreach (#PCDATA)>
<bw:!ATTLIST foreach
  element  %ElemName;    #REQUIRED
  autodiv  %BlockName;   #IMPLIED
  maxn     %Count;       #IMPLIED
>

<bw:!--=========== Group Element =============-->
<bw:!ELEMENT group (#PCDATA)>

<bw:!--=========== Conditional Element =============-->
<bw:!ELEMENT if (#PCDATA)>
<bw:!ATTLIST if
  expr     %Expression;  #REQUIRED
>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:!-- ========== Ignore Element ============-->
<bw:!ELEMENT ig (#PCDATA)>

<bw:!-- ========== Include Element ============-->
<bw:!ELEMENT include (#PCDATA)>
<bw:!ATTLIST include
  src      %FileName;    #REQUIRED
>

<bw:!-- ========== Variable Element ============-->
<bw:!ELEMENT var (#PCDATA)>
<bw:!ATTLIST var
  name     %VarName;     #IMPLIED
  expr     %Expression;  #REQUIRED
>

<bw:!-- ========== Use Class Element ============-->
<bw:!ELEMENT usecls (#PCDATA)>
<bw:!ATTLIST usecls
  name     %ClsName;     #REQUIRED
>

<bw:!-- ========== Lookup Table Elements ============-->
<bw:!ELEMENT lutab (te)>
<bw:!ATTLIST
  name     CDATA         #REQUIRED
>

<bw:!ELEMENT te EMPTY>
<bw:!ATTLIST te
  key      CDATA         #REQUIRED
  val1     CDATA         #REQUIRED
  val2     CDATA         #IMPLIED
  val3     CDATA         #IMPLIED
  val4     CDATA         #IMPLIED
  val5     CDATA         #IMPLIED
>

<bw:!ELEMENT lookup EMPTY>
<bw:!ATTLIST lookup
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  name     CDATA         #REQUIRED
  element  CDATA         #IMPLIED
  inlist   CDATA         #REQUIRED
  key      CDATA         #IMPLIED
>

<bw:!--=========== Script Element =============-->
<bw:!ELEMENT script (#PCDATA)>
<bw:!ATTLIST script
  language CDATA         #REQUIRED
  type     CDATA         #IMPLIED
  src      CDATA         #IMPLIED
>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Anhang C

# BML - Formale Definition

Expression-Syntax (ohne Widgets):

```
Expr     ::= Expr InfixOp Expr | FuncName Args | RefExpr | '(' Expr ')'
|
             PräfixOp Expr.
Args     ::= '(' ')' | '(' Expr {',' Expr} ')'.
PräfixOp ::= '-' | 'not'.
InfixOp  ::= 'and' | 'or' | '+' | '-' | '*' | '%' | '/' | '=' | '!=' |
             'eq' | 'ge' | 'gt' | 'le' | 'lt' | 'ne'.
RefExpr  ::= Identifier ['[' Expr ']'] {RefOp RefExpr}.
RefOp    ::= '.' | '.*'.

Eine RefExpr kann statt mit einem Identifier auch mit einem '.' begin-
nen; dies ist gleichbedeutend mit dem aktuellen Element einer umgebenden
for-each-Schleife. Der Ausdruck 'pos()' bezeichnet den aktuellen Index
(beginnend mit 1) einer umgebenden foreach-Schleife.
```

Operator-Präzedenz:

| Präzedenz | Operator |
|---|---|
| 0 | * / % div |
| 1 | + - |
| 2 | = != eq ge gt le lt ne |
| 3 | and or |

Bisherige Basis-Methoden (alle public-Methoden in biss.BML.Evaluator):

```
[Derzeit müssen alle Methoden String-Argumente haben. Dabei ist zu be-
achten, dass sämtliche Vergleichsoperationen ihre Argumente numerisch
vergleichen. Ein Stringvergleich erfolgt mit der Methode ,compare(...)']
```

String-Methoden:

```
public int compare ( String s1, String s2)
public String concat ( String s1, String s2)
```



© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
public String concat ( String s1, String s2, String s3)
public String concat ( String s1, String s2, String s3, String s4)
public String concat ( String s1, String s2, String s3, String s4,
                       String s5)
public boolean endsWith ( String s1, String s2)
public int indexOf ( String s1, String s2)
public String join ( String delim, String s1, String s2)
public String join ( String delim, String s1, String s2, String s3)
public String join ( String delim, String s1, String s2, String s3,
                     String s4)
public String join ( String delim, String s1, String s2, String s3,
                     String s4, String s5)
public int lastIndexOf ( String s1, String s2)
public boolean startsWith ( String s1, String s2)
public String substring ( String s1, String s2, String s3)

Numerische Methoden:
public String mod ( String s1, String s2)
public String mod2 ( String s1)
public String sgn ( String s1)


Spezielle Methoden zur optionalen Erzeugung von Commands:

public String cmd ( String name)
public String cmd ( String name, String p1)
public String cmd ( String name, String p1, String p2)
public String cmd ( String name, String p1, String p2, String p3)
public String cmd ( String name, String p1, String p2, String p3,
                    String p4)
public String cmd ( String name, String p1, String p2, String p3,
                    String p4, String p5)
public String cmdIg ( String name)
public String cmdIg ( String name, String p1)
public String cmdIg ( String name, String p1, String p2)
public String cmdIg ( String name, String p1, String p2, String p3)
public String cmdIg ( String name, String p1, String p2, String p3,
                      String p4)
public String cmdIg ( String name, String p1, String p2, String p3,
                      String p4, String p5)
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

Beispiel einer spezifischen Methode in einer von biss.BML.Evaluator abgeleiteten Klasse):

```
public String fullName ( String geschlecht, String titel, String vorname,
                         String nachname)
{
   return join( " ", ("weiblich".equals( geschlecht) ? "Frau" : "Herr"),
                    titel, vorname, nachname);
}
```

Zugriff auf das DOM (Beispiel in einer von biss.BML.Evaluator abgeleiteten Klasse):

```
public String fullName ( String partnerPersAcc)
{
   Object partnerPers = getItem( null, partnerPersAcc);
   Object partner = getItem( partnerPers, "Partner");
   String geschlecht = getField( partnerPers, "Geschlecht");
   String titel = getField( partnerPers, "Titel");
   String vorname = getField( partner, "Name1");
   String nachname = getField( partner, "Name2");
   return join( " ", ("weiblich".equals( geschlecht) ? "Frau" : "Herr"),
                    titel, vorname, nachname);
}
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

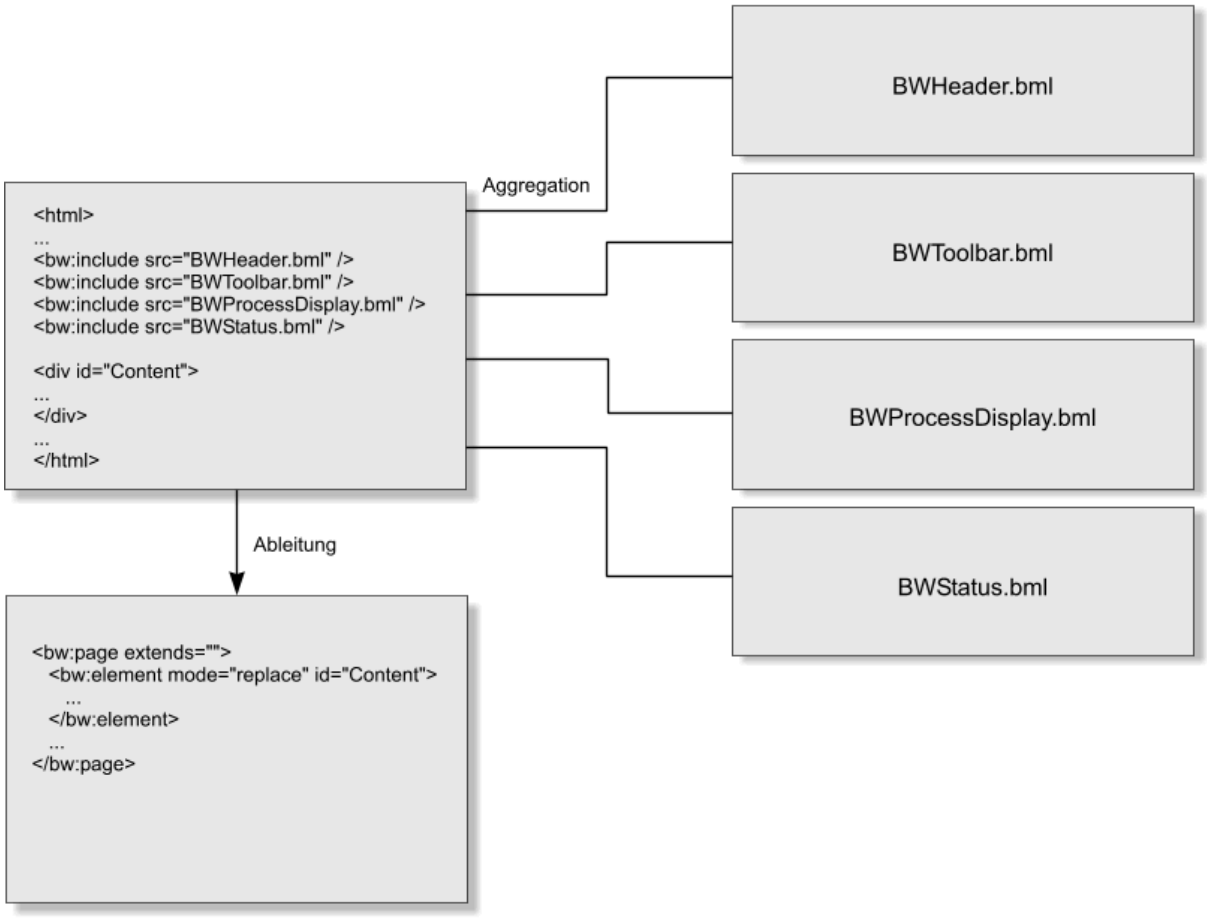
Anhang D

# Beispiel-Templatedatei

## Beispiel-Templatedatei (BWisePortal.BML)

Die folgenden Seiten zeigen eine recht komplexe BML-Datei, wie sie innerhalb von *B-Wise* benutzt wird. Es handelt sich dabei um das *BWisePortal*, also die Einstiegsseite der *B-Wise*-Applikation.

Die Struktur der BWisePortal-Seiten sieht wie folgt aus:




© Copyright BISS GmbH, 2001-2012 All Rights reserved

Alle *B-Wise*-Seiten basieren auf der Basisseite BWBase.BML. Diese enthält ausschließlich die Basis-Layout-Informationen und inkludiert die Standard-Elemente Header, Toolbar, Prozessanzeige sowie Statuszeile. Die Basisseite enthält zusätzlich die Definition der Bereiche für Struktur-Informationen (Structure) sowie den Inhalt (Content), die jedoch leer sind und erst in den abgeleiteten Seiten definiert werden.

Jede weitere Seite ist von BWBase abgeleitet und definiert lediglich den Struktur- sowie den Inhaltsbereich. Ggf. kann die Toolbar überschrieben oder ergänzt werden.

Die Ableitungshierarchie kann weiter fortgesetzt werden; so wird z.B. eine Seite BWPolicyDisplay.BML (zur Vertragsanzeige) definiert, welche die Basis für alle Sparten enthält; die Sparten-spezifischen Anteile werden anschließend über eine weitere Ableitung modelliert (hier wäre allerdings ebenfalls eine Aggregation mit Hilfe eines `<bw:include>` möglich).

Die folgenden Quellcode-Beispiele enthalten die o.g. Seiten.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Anhang E

# *B-Wise* SmartClient-Einstellungen

Eine Reihe von Layout-Definitionen und Einstellungen für den SmartClient sind in Dateien mit der Dateierweiterung „.config“ untergebracht. Diese befinden sich im `bin`-Verzeichnis des SmartClients. Die meisten dieser Dateien können vom Administrator oder sogar vom Endanwender an die Unternehmens- bzw. individuellen Bedürfnisse adaptiert werden. Bitte beacten Sie, dass ungültige oder gar syntaktisch falsche Änderungen zu Fehlern bis hin zum Nicht-Funktionieren des Clients führen können.

Folgende Konfigurationsdateien stehen im SmartClient zur Verfügung (in Zweifelsfällen ist die im jeweiligen Projekt-Repository eingechekcte Datei gültig!):

## BWSmartClientCore_Com.config

Diese Konfigurationsdatei definiert einige Kommunikationsattribute sowie Logging-Möglichkeiten. Die für den Administrator „zugänglichen“ Elemente sind fett hervorgehoben.

Das Element `<bw:timeout>` deifniert diejenige Zeit (in msec), die der Client auf eine Antwort vom *B-Wise* Server wartet, bevor die Session seitens des Clients abgebrochen wird.

Die Elemente des Log-Bereichs ermöglichen eine mehr oder weniger detaillierte Aufzeichnung der Kommunikation zwischen Client und Server. Wird eine Protokollierung (Logging) gewünscht, ist in das Element `<bw:toFile>` ein gültiger Dateiname (optional inkl. Pfad) einzutragen. Zusätzlich kann mit Hilfe der Elemente `<bw:unknownElems>` und `<bw:unknownAttrs>` bestimmt werden, ob auch in BML nicht definierte Elemente bzw. nicht definierte Attribute als Fehler protokolliert werden sollen. Dies hilft insbesondere bei der Fehlersuche im Falle von „Vertippern“.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:config xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:xmlrpc>
    <bw:client>SmartClient</bw:client>
    <bw:encoding>ISO-8859-1</bw:encoding>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      <bw:timeout>100000</bw:timeout>
      <bw:keepAlive>false</bw:keepAlive>
   </bw:xmlrpc>

   <bw:log>
      <bw:unknownElems>false</bw:unknownElems>
      <bw:unknownAttrs>false</bw:unknownAttrs>
      <bw:toFile>LogFileName.log</bw:toFile>
   <bw:log>

   <w:update>
      <bw:ignore>false</bw:ignore>
   </bw:update>

</bw:config >
```

# BWSmartClientCore_Portal.config

Diese Konfigurationsdatei enthält primär Informationen zum grundsätzlichen Layout des Portals bzw. der Portlets.

Alle Bereiche können vom Administrator oder Endanwender geändert und an individuelle Wünsche angepasst werden. Die Bedeutung der einzelnen Element ist den Kommentaren innerhalb des nachfolgend aufgeführten Beispiels zu entnehmen.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:config xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

   <!-- Definiert die Breite des Splitbar zwischen zwei Panes und deren Farbe;
      Alle Farben sind als Hexadezimal-Werte in der Notation #rrggbb anzugeben.
      Zusätzlich wird der Rand zwischen Pane und Portlet definiert -->
   <bw:portal>
      <bw:splitbar>
         <bw:width>3</bw:width>
         <bw:color>transparent</bw:color>
      </bw:splitbar>
      <bw:margin>1</bw:margin>
   </bw:portal>

   <!-- Definiert das Layout aller Portlets -->
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
<bw:portlet>
   <!-- Farbe und Breite der Portlet-Umrahmung -->
   <bw:frame>
      <bw:color>#555555</bw:color>
      <bw:width>1</bw:width>
   </bw:frame>

   <!-- Farbe und Schrift der Portlet-Titlebar -->
   <bw:titlebar>
      <!-- Hintergrund-Farbe; alternativ zum Element <bw:color> kann auch das
         Element <bw:brush> angegeben werden, welches dann einen
         Farbverlauf von der Farbe <bw:c0> zur Farbe <bw:c1> definiert. -->
      <bw:background>
         <bw:brush>
            <bw:c0>#D8E7F6</bw:c0>
            <bw:c1>#84B5E6</bw:c1>
         </bw:brush>
      </bw:background>
      <bw:foreground>
         <bw:color>black</bw:color>
      </bw:foreground>
      <!-- Definiert die Schriftart des Titel-Textes -->
      <bw:font>Microsoft Sans Serif,8,Regular</bw:font>
   </bw:titlebar>

   <!-- Definiert die Hintergrundfarbe eines Portlets -->
   <bw:background>
      <bw:color>#E0E0E0</bw:color>

      <!-- Alternative Möglichkeiten mit dem <bw:brush>-Element:
      <bw:brush>
         <bw:c0>Red</bw:c0>
         <bw:c1>Blue</bw:c1>
         <bw:width>100</bw:width>
         <bw:height>100</bw:height>
      </bw:brush
      -->
   </bw:background>

   <!-- Definiert optionale Bedien-Icons innerhalb der Titlebar sowie die
      Transparenz dieser Icons -->
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
    <bw:toolstrip>
      <bw:system>
        <bw:transparency>0</bw:transparency>
        <bw:reloadable>true</bw:reloadable>
        <bw:sizeable>true</bw:sizeable>
        <bw:closeable>true</bw:closeable>
      </bw:system>
    </bw:toolstrip>
  </bw:portlet>

</bw:config >
```

## BWSmartClientApp_User.config

Diese Konfigurationsdatei enthält die Position und Größe des SmartClient-Hauptfensters sowie optional den Benutzernamen und das Passwort (Login). Diese Datei ist normalerweise verschlüsselt und wird automatisch geschrieben, wenn der Endanwender die Checkbox „Login-Daten lokal speichern“ und/oder „Fensterposition speichern“ wählt.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:config xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:pos aox="0" aoy="0" acx="0" acy="0" eox="0" eoy="0" ecx="1024" ecy="768"/>
  <bw:user></bw:user>
  <bw:password></bw:password>

</bw:config >
```

## BWSmartClientApp_Menu.config

Diese Konfigurationsdatei enthält die gesamte Definition des initialen Menüs sowie der initialen Toolbar des SmartClients.

Sowohl Menü als auch Toolbar können im laufenden Betrieb von *B-Wise* Prozessen geändert (erweitert) werden. Die in dieser Konfigurationsdatei definierten Enträge sind jedoch fix und bleiben während der gesamten Ausführungsdauer des SmartClients unverändert (sie können jedoch von *B-Wise* Prozessen „disabled“, d.h. auf inaktiv gesetzt werden).


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Ist diese Konfigurationsdatei vorhanden, werden sowohl das Menü als auch die Toolbar des SmartClients ausschließlich mit Hilfe dieser Datei definiert. Ist die Datei nicht vorhanden, verfügt der SmartClient über ein vordefiniertes Menü sowie eine vordefinierte Toolbar.

Der folgende Code enthält ein mögliches Beispiel. Bei den einzelnen Menüpunkten kann anstatt des `imgRes`-Attributs auch das `img`-Attribut verwendet werden. Während das `imgRes`-Attribut die einzelnen Images aus der Ressource-Datei liest, nutzt das `img`-Attribut einzelne Icon-Dateien, die im GIF-, BMP- oder PNG-Format vorliegen können und die Abmessungen 16x16 Pixel haben müssen.

Bezüglich der möglichen Funktionen (also Inhalte des `action`-Attributs) wird auf die Funktionalität des jeweiligen *B-Wise* Servers verwiesen.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:config xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:dlgmenu>
    <bw:menu>
      <bw:msub label="B-Wise">
        <bw:mi label="Prozess~liste" accel="C+L"
             imgRes="BWSmartClientApp.List"/>
        <bw:msub id="procs" label="Prozesse" imgRes="BWSmartClientApp.Cog"/>
        <bw:msep/>
        <bw:mi label="Beenden" accel="A+F4" action="syscmd_exit"/>
      </bw:msub>
      <bw:msub label="Bearbeiten">
        <bw:mi label="Suchen" accel="C+F" action="B_SRC()"
                                          imgRes="BWSmartClientApp.Search"/>
        <bw:mi label="Drucken" accel="C+D" action="B_PRN()"
             imgRes="BWSmartClientApp.Printer"/>
        <bw:mi label="Speichern" action="B_SAV()" accel="C+S"
             imgRes="BWSmartClientApp.Save"/>
        <bw:msep/>
        <bw:mi label="Beenden/Abbrechen" accel="C+A" action="B_CAN()"
             imgRes="BWSmartClientApp.Browser"/>
      </bw:msub>
      <bw:msub label="Ansicht">
        <bw:mci label="~Prozesse" checked="true" accel="C+P"
             action="syscmd_showProcs(@checked)"/>
        <bw:msep/>
        <bw:msub label="Portlets" id="portlets"
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
            imgRes="BWSmartClientApp.Portlets"/>
      <bw:msep/>
      <bw:mi label="Fensterpositionen speichern" accel="C+W"
             action="syscmd_saveDlgBounds" imgRes="BWSmartClientApp.SavePos"/>
    </bw:msub>
    <bw:msub label="Hilfe">
      <bw:mi label="Hilfe-Startseite" accel="C+F1"
            imgRes="BWSmartClientApp.Help" action="syscmd_helpMain"/>
      <bw:mi label="Benutzerhandbuch" accel="A+F1"
            imgRes="BWSmartClientApp.Book" action="syscmd_helpBook"/>
      <bw:mi label="Dialoghilfe" accel="F1" action="syscmd_helpDialog"/>
      <bw:msep/>
      <bw:mi label="Client-Info" action="syscmd_helpClient"/>
      <bw:mi label="Server-Info" action="syscmd_helpServer"/>
    </bw:msub>
  </bw:menu>
</bw:dlgmenu>

<bw:dlgtoolbar>
  <bw:tbcontainer>
    <bw:tbbutton id="tbBack" action="B_PRV()">
      <bw:img res="BWSmartClientApp.Back"/>
      <bw:val>Zurück</bw:val>
      <bw:tooltip>
        <bw:ttb>Zurück zum vorherigen Schritt oder Prozess</bw:ttb>
      </bw:tooltip>
    </bw:tbbutton>
    <bw:tbseparator/>
    <bw:tbbutton id="tbCancel" action="B_CAN()">
      <bw:img res="BWSmartClientApp.Browser"/>
      <bw:val>Beenden</bw:val>
      <bw:tooltip>
        <bw:ttb>Aktuellen Prozess abbrechen</bw:ttb>
      </bw:tooltip>
    </bw:tbbutton>
    <bw:tbbutton id="tbHome" action="F_POR()">
      <bw:img res="BWSmartClientApp.Home"/>
      <bw:tooltip>
        <bw:ttb>Zum B-Wise Portal wechseln</bw:ttb>
      </bw:tooltip>
    </bw:tbbutton>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      <bw:tbbutton id="tbReload" action="HTML_RELOAD()">
        <bw:img res="BWSmartClientApp.Refresh"/>
        <bw:tooltip>
          <bw:ttb>Aktuellen Prozess initialisieren</bw:ttb>
        </bw:tooltip>
      </bw:tbbutton>
      <bw:tbseparator/>
      <bw:tbbutton id="tbProcs" action="F_PRL()">
        <bw:img res="BWSmartClientApp.List"/>
        <bw:tooltip>
          <bw:ttb>Liste aller Prozesse anzeigen</bw:ttb>
        </bw:tooltip>
      </bw:tbbutton>
      <bw:tbseparator/>
      <bw:tbbutton id="tbPrint" action="B_PRN()">
        <bw:img res="BWSmartClientApp.Printer"/>
        <bw:tooltip>
          <bw:ttb>Aktuelle Ansicht oder das aktuelle Dokument drucken</bw:ttb>
        </bw:tooltip>
      </bw:tbbutton>
      <bw:tbbutton id="tbSearch" action="B_SRC()">
        <bw:img res="BWSmartClientApp.Search"/>
        <bw:tooltip>
          <bw:ttb>Kunden-Suchprozess starten</bw:ttb>
        </bw:tooltip>
      </bw:tbbutton>
      <bw:tbseparator/>
      <bw:tbchoice id="tcPsearch" align="right">
      </bw:tbchoice>
      <bw:tbbutton id="tbPsearch" align="right" passoc="tcPsearch"
             action="QuickSearch(@Text)">
        <bw:img res="BWSmartClientApp.User"/>
        <bw:tooltip>
          <bw:ttb>Kunden-Suchprozess mit Suchmuster starten</bw:ttb>
        </bw:tooltip>
      </bw:tbbutton>
    </bw:tbcontainer>
  </bw:dlgtoolbar>

</bw:config>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

# BWSmartClientApp_Login.config

Diese Konfigurationsdatei enthält Einstellungen des Login-Bildschirms, d.h. Inhalte der einzelnen Controls sowie das Layout der Darstellung. Die Bedeutung ist in den folgenden Kommentaren wiedergegeben.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:config xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <!-- Inhalt der Sprachauswahl-Choice; die id's entsprechen der internationalen
     Kodierung; es ist zu beachten, dass nur solche Sprachen angeboten werden
     sollten, die der B-Wise Server auch unterstützt -->
  <bw:languages>
     <bw:language id="" display="Benutzerabhängig"/>
     <bw:language id="DE" display="Deutsch (D)"/>
     <bw:language id="DE_AT" display="Deutsch (A)"/>
     <bw:language id="EN_US" display="Englisch (USA)"/>
     <bw:language id="EN_UK" display="Englisch (GB)"/>
     <bw:language id="NL" display="Holländisch"/>
     <bw:language id="CZ" display="Tschechisch"/>
  </bw:languages>

  <!-- Liste der verfügbaren B-Wise Server -->
  <bw:hosts>
     <bw:host>http://localhost:3083</bw:host>
     <bw:host>http://pizza.biss-net.com:3083</bw:host>
  </bw:hosts>

  <!-- Hintergrund-farben des Login-Dialogs -->
  <bw:view>
     <!-- Farbe der beiden Balken oben und unten -->
     <bw:headcolor>#95A6A6</bw:headcolor>
     <!-- Farbverlauf zwischen den Balken von oben (<bw:gcolor0>) nach unten
        (<bw:gcolor1>) -->
     <bw:gcolor0>white</bw:gcolor0>
     <bw:gcolor1>#A3B7B8</bw:gcolor1>
  </bw:view>

  <!-- Angabe, ob die AboutBox beim Starten angezeigt werden soll -->
  <bw:startup>
     <bw:splashscreen>true</bw:splashscreen>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
  </bw:startup>

</bw:config>
```

# BWSmartClientApp_Help.config

Diese Konfigurationsdatei enthält die Hilfetexte, die innerhalb des Login-Dialogs angezeigt werden können. Diese Hilfetexte beziehen sich auf die einzelnen Eingabe- bzw. Auswahlfelder und werden angezeigt, sobal das entsprechende Control den Fokus besitzt.

Die hier definierten Texte werden verwendet, wenn der SmartClient diese Datei beim Start innerhalb seines `bin`-Verzeichnisses vorfindet. Ist die nicht der Fall verfügt der SmartClient über entsprechende Standard-Texte.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:config xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

  <bw:help>
    <bw:field name="anmeldung">
      Bitte melden sich sich mit den unten stehenden Daten am System an.
      Weiterghende Hilfe zu jedem Feld erhalten Sie durch Click auf den Feldtext
      (Prompt) bzw. wenn sich der Cursor (Focus) im entsprechenden Feld befindet.
    </bw:field>
    <bw:field name="benutzername">
      Bitte geben Sie hier den Ihnen zugewiesenen Benutzernamen (das sog.
      "Login") ein. Beachten Sie bitte, dass zwischen Groß- und Kleinschreibung
      unterschieden   wird !
    </bw:field>
    <bw:field name="passwort">
      Bitte geben Sie hier das Ihnen zugewiesene oder von Ihnen selbst gewählte
      Passwort ein. Besitzen Sie lediglich ein Ihnen zugewiesenes Passwort, geben
      Sie es hier ein; Sie können es später in ein Ihnen genehmes Passwort
      ändern. Beachten Sie bitte auch hier, dass zwischen Groß- und
      Kleinschreibung unterschieden wird !
    </bw:field>
    <bw:field name="servername">
      Der Servername bestimmt denjenigen Server, mit dem Sie gerade arbeiten
      möchten. Durch Auswahl des gewünschten Servers bestimmen Sie vornehmlich
      die Funktionalität, die Ihnen anschließend zur Verfügung steht sowie die
      Daten (Bestands- und/oder Akquisedaten), mit denen Sie arbeiten können. Der
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
     "lokale Server" ist die auf Ihrem persönlichen System installierte Version
     mit den zugehörigen Daten; diesen Server können Sie auch ohne eine
     Netzwerk-Verbindung (z.B. zum Internet) verwenden. Sollte ein bestimmter
     Server nicht verfügbar sein, wenden Sie sich bitte an den Support.
    </bw:field>
    <bw:field name="sprache">
     Aktuell unterstützt das System ausschließlich die Deutsche Sprache; künftig
     werden auch andere, länder-spezifische Versionen verfügbar sein.
    </bw:field>
    <bw:field name="token">
     Manche Produktanbieter verlangen für eine korrekte Anmeldung ein sog
     "Token". Dabei handelt es sich um eine mehrstellige Zeichenfolge
     (normalerweise eine Zahl), die mit Hilfe eines kleinen Schlüsselanhänger-
     ähnlichen Gerätes erzeugt wird. Sind Sie im Besitz eines solchen Gerätes,
     dann geben Sie das vom Gerät erzeugte Token (Zahl) hier ein. Bitte beachten
     Sie, dass Sie ohne Eingabe des Tokens nicht mit denjenigen Anbietern
     zusammen arbeiten können, die ein solches Token verlangen. Weitere Hinweise
     hierzu bzw. Hinweise, wie Sie an ein solches Token gelangen, können Sie
     beim Support erfragen.
    </bw:field>
  </bw:help>

</bw:config>
```

# BWSmartClientApp_Warning.config

Die folgende Konfigurationsdatei enthält alle verfügbaren Warnhinweise.

Die hier definierten Texte werden verwendet, wenn der SmartClient diese Datei beim Start innerhalb seines `bin`-Verzeichnisses vorfindet. Ist die nicht der Fall verfügt der SmartClient über entsprechende Standard-Texte.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bw:config xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

<bw:warnings>
   <bw:warning name="serverBusy">
     Sie sind im Begriff die Anwendung zu schließen obwohl der Server
     noch einen Prozess ausführt. Dies kann zu einem Datenverlust führen.
     Drücken Sie daher bitte die Schaltfläche "Retry" und warten je nach
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
      ausgeführtem Prozes ggf. einige Sekunden.

      Sollte der Server auf Ihre Eingaben nicht mehr reagieren, ist dies
      allerdings die einzige Möglichkeit, ihn zu beenden. Tritt dieses
      Problem häufiger auf, wenden Sie sich bitte an den Support.
   </bw:warning>
</bw:warnings>

</bw:config>
```

# BWSmartClientApp_Error.config

Die folgende Konfigurationsdatei enthält alle verfügbaren Fehlermeldungen.

Die hier definierten Texte werden verwendet, wenn der SmartClient diese Datei beim Start innerhalb seines `bin`-Verzeichnisses vorfindet. Ist die nicht der Fall verfügt der SmartClient über entsprechende Standard-Texte.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bw:config
   xmlns:bw="http://www.biss-net.com/2001/bwise/ehtml">

<bw:errors>
   <bw:error name="capsdenied">
      Die von Ihnen eingegebene Kombination aus Benutzername und Passwort
      ist falsch. Sie haben (aus Versehen ?) die Umschalt-Feststell-Taste
      gedrückt. Bitte schalten Sie wieder auf Groß-/Kleinschreibung um und
      versuchen es noch einmal.
   </bw:error>
   <bw:error name="denied">
      Die von Ihnen eingegebene Kombination aus Benutzername und Passwort
      ist falsch.
   </bw:error>
   <bw:error name="noserver">
      Der von Ihnen gewählte Server ist nicht verfügbar. Handelt es sich
      dabei nicht um den lokalen Server, wenden Sie sich bitte an den
      Support. Sollte diese Fehlermeldung  bei Auswahl des lokalen Servers
      angezeigt werden, liegt ein technisches Problem mit Ihrem System vor;
      wenden Sie sich hierzu an den Anbieter Ihres Verwaltungsprogramms.
   </bw:error>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
   <bw:error name="sessiondied">
      Die Verbindung wurde aus unbekannten Gründen vom Server unterbrochen.
   </bw:error>
</bw:errors>

</bw:config>
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Anhang F

# Attribut-Evaluation

Folgende BML-Elemente enthalten Attribute, die automatisch (d.h. ohne die Verwendung von geschweiften Klammern) evaluiert werden. Die Ellipsis (drei Punkte) deutet an, dass dies für alle Elemente gilt, bei denen das entsprechende Attribut gültig ist.

In der folgenden Aufstellung bedeutet `const`, dass das Attribut ohne geschweifte Klammern nicht evaluiert wird, `expr` hingegen, dass das Attribut auch ohne geschweifte Klammern evaluiert wird:

```
<bw:case value="expr" />
<bw:copy element="expr" dropRoot="expr" exclude="const"
          transform="expr" />
<bw:element mode="const" path="expr" id="const" pos="expr" />
<bw:foreach element="expr" startpos="expr" maxn="expr"
          autodiv="const" />
<... keep="expr" .../>
<bw:if expr="expr" value="const">
<bw:language name="expr" />
<script .. content="expr" />
<bw:switch expr="expr" />
<... datasrc="expr" .../>
<... element="expr" .../>
```

Soll daher innerhalb eines Attributs, welches automatisch evaluiert wird, eine Konstante verwendet werden, ist die in einfache Hochkommata zu setzen. Beispiel:

```
<bw:if expr="'true'">
```

Bei allen anderen Attributen müssen geschweifte Klammern verwendet werden, um den darin enthaltenen Ausdruck vom BML-Transformer evaluieren zu lassen. Beispiel:

```
<div id="{blk_id}">
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

BISS

Anhang G

# BML-Dialogelemente

Die folgende Tabelle fasst die BML-Dialogelemente, deren Attribute, API-Funktionen sowie Events zusammen.

Die Tabelle ist wie folgt aufgebaut:

In der ersten Spalte ist der technische (*B-Wise* interne) Name des Elements aufgeführt. Dabei handelt es sich um den im SmartClient (oder ggf. auch im BrowserClient) definierten Klassennamen. In Klammern darunter steht die zugehörige Basisklasse, so dass ersichtlich ist, welche Attribute, Commands etc. das jeweilige Element von seiner Basisklasse erbt.

In der zweiten Spalte steht der Element (Tag-) Name ohne den „`bw`"-Namensraum. Innerhalb einer BML-Datei ergibt sich der komplette Elementname somit aus dem Namensraum „`bw`" und dem hier stehenden Namen, eingeschlossen in spitze Klammern, z.B. „`<bw:button>`".

Die dritte Spalte enthält alle erlaubten Attribute; beachten Sie, dass hier auch die Attribute der Basisklasse dazu gehören. So kann das Element „`button`" nicht nur das eigene Attribut „`def`" enthalten, sondern ebenfalls alle Attribute der Basisklasse `BWValCtrl` und hier wiederum von dessen Basisklasse, also `BWCtrl` (`state`, `tabindex`, `enabled`, `visible`). Der Typ des Attributs wird in Klammern angegeben; dabei bedeuten „S" = String, „E" = Enumeration (Aufzählung, die erlaubten Werte stehen unmittelbar dahinter), „B" = Bool (hier können die Werte „true" oder „false" angegeben werden oder ein Ausdruck, der zu einem dieser beiden Werte evaluiert), „I" = Interger (Ganze Zahl).

Die vierte Spalte enthält alle erlaubten „inner tags", also Elemente, die innerhalb des aufgeführten Elements zulässig sind. Auch hier ist die Klassenhierarchie zu beachten, d.h. das „`button`"-Element kann nicht nur das `<img>`-Element enthalten sondern auch die unter `BWValCtrl` und `BWCtrl` aufgeführten „inner tags".

Die fünfte Spalte dokumentiert das Server-seitige API, d.h. diejenigen Methoden, die das Element versteht und die vom Server aus (bzw. aus *B-Wise* Prozessen) an das Element gesendet werden können. So versteht der Button die Methode „`setDefault`" sowie alle Methoden, die unter `BWValCtrl` und `BWCtrl` aufgeführt sind.


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Die sechste Spalte führt alle Events auf, die vom Element getriggert werden und die siebte Spalte enthält die zugehörigen Event-Parameter. Diese können im Rahmen der abzuarbeitenden Methode dieser mitgegeben werden. Der folgende BML-Code definiert ein mögliches Beispiel:

```
<bw:list ...>
...
<bw:event
  name="selected"
  method="{concat( 'handleEvent( edit,', concat( PID, ',','@nid',')'))}"
/>
...
</bw:list>
```

Hier wird (bei einer Liste) im Falle der Auswahl einer Zeile durch den Anwender die Prozess-Methode “edit” mit den Parametern PID (Prozess-ID) und “nid” (Node-ID) des selektierten Elements aufgerufen. Um den Prozessaufruf korrekt zu gestalten, werden die einzelnen Elemente (Methodenname und Parameter) mit Hilfe der BML-Funktion `concat` zusammengestellt.

[Die Tabelle ist eine externe Datei…]

© Copyright BISS GmbH, 2001-2012 All Rights reserved

BISS

Anhang H

# *B-Wise* Namenskonventionen

*Namen sind wichtig ! Gute Namen sind schwer zu finden !*

Die innerhalb von *B-Wise* verwendeten Namen sollten möglichst kurz sein, trotzdem aber genügend semantische Informationen enthalten. Dies ist insbesondere bei den IDs, also den eindeutigen Identifikatoren der BML-Elemente wesentlich.

Neben dieser eher allgemeinen Regel, existieren in *B-Wise* folgende Namenskonventionen für die Definition von IDs:

- ID-Namen sind case-insensitive, d.h. Groß- und Kleinschreibung spielen keine Rolle und sollten daher zur Verbesserung der Lesbarkeit verwendet werden;
- Jedes Element hat einen eindeutigen Präfix, der jeweils bei der Beschreibung des Elements aufgeführt ist. So hat z.B. ein `<bw:label>` den Präfix „LA_“, d.h. alle IDs eines `<bw_label>`-Elements sollten mit „LA_“ beginnen, z.B. „LA_Name“;
- Sind zwei Elemente unterschiedlichen Typs logisch assoziiert, sollten sie denselben ID-Namen tragen und sich lediglich durch den Präfix unterscheiden. Werden auf einem Dialog z.B. für die Erfassung eines Nachnamens ein Label und ein Entryfield verwendet, so sollten diese wie folgt benannt werden: „LA_Name“ für das `<bw:label>` und „TF_Name“ für das assoziierte `<bw:entry>`;
- Namen werden grundsätzlich zusammengeschrieben; Leerzeichen sind nicht erlaubt und es sollten auch keine Unterstriche - mit Ausnahme des Präfix - verwendet werden. Logisch getrennte Worte können durch Großbuchstaben gekennzeichnet werden, z.B. „TF_GebDat“ für ein Geburtsdatum-Eingabefeld;
- Es sollten gängige Abkürzungen verwendet werden; so sollte z.B. die Versicherungsscheinnummer als VSNR abgekürzt werden; ebenso sind „langer Ausdrücke“ wenn möglich sinnvoll abzukürzen (z.B. anstatt „TF_HaftpflichtJahresBeitrag“ besser bzw. „TF_HPJBeitrag“).

**Konkrete Konventionen müssen im Projekt festgelegt werden!**


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Anhang J

# Versionsinformationen

Der folgende Abschnitt enthält die Versionsinformationen zur Fortentwicklung der *B-Wise* BML. Die Versionsinformationen sind chronologisch geordnet, wobei die neueste Information zuerst aufgeführt ist. Die aufgeführten Informationen enthalten auch technische Anmerkungen und Hinweise zur Implementierung; diese sind jedoch ggf. auch für den BML-Designer hilfreich und wurden daher in diese Versionsinformationen übernommen.

**Achtung:** Die hier genannten Versionsnummern sind die *B-Wise* internen Versionsnummern des BML-Moduls und stimmen nicht mit den offiziellen *B-Wise* Versionsnummern überein !

### Version 2.0.3

- 'Parameter'-Variablen in `<bw:include>`; `<bw:include>` macht nun implizit immer einen eigenen Scope auf (implizites `<bw:scope>`). Im `<bw:include>` deklarierte Variablen bekommen nun automatisch das Attribut `local="true"`, falls dieses Attribut nicht schon vorhanden ist (ihr Scope endet damit mit dem `<bw:include>`). Damit verhalten sich solche Variablen wie Parameter für die inkludierte Datei.

### Version 2.0.2

- Unterstützung anonymer (unbenannter) Listen-Elemente: In Listen Ausdrücken kann nun der Listen-Element-Name weggelassen werden Beispiel "Adressen.*.Strasse" anstelle von "Adressen.*Adresse.Strasse". Im DOM-Kontext war dieser Name sowieso irrelevant, im XML-Kontext können damit jetzt alle Subelemente erfasst werden.
- Neue Methode 'isEmpty' prüft, ob ein XML Element leer ist (keine Subelemente und keinen Textinhalt enthält).

### Version 2.0.1

- Neue Methode 'textContent' liefert den Inhalt (Textcontent + TextContent aller Subelemente) eines XML Elementes, welches als Parameter übergeben wird.
- Neues 'exclude'-Attribut von bw:copy; damit kann eine Liste von Elementnamen angegeben werden, welche nicht kopiert werden sollen.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Version 2.0

- Umstellung auf JDK 1.5
- Umstellung auf org.w3c.dom; JDOM und ICE-DOM wird nicht mehr unterstützt.
- Unterstützung von Internationalisierung: `<bw:t>`-Elemente und `t()`-Methoden werden übersetzt, falls im Transformer ein LocalizationContext gesetzt ist.
  `<bw:language name="languageId" > ... </bw:language>` setzt die Ursprungssprache innerhalb dieses Blocks; dieses Konstrukt kann auch verschachtelt verwendet werden.

### Version 1.6.3

- Unterstützung von Scoped-Variables; mit Hilfe des neuen Tag `<bw:scope>...</bw:scope>` kann ein neuer Scope aufgemacht; Variablen (`<bw:var>`) können jetzt das zusätzliche Attribut `local="true"` erhalten; eine solche Variable ersetzt eine vorher existierende Variable gleichen Namens. Am Ende des Scopes werden alle Variablen restauriert, welche durch innerhalb des aktuellen Scopes definierte lokale Variablen ersetzt wurden.

### Version 1.6.2

Neue Methode `'xmlContent'` liefert den Inhalt (Subelemente + Textcontent) eines XML Elementes, welches als Parameter übergeben wird.

### Version 1.6.1

- Neue Methode 'action' mit den selben Parametern wie 'cmd': liefert nur den reinen Link.
- Neue Methode 'grant' mit den selben Parametern wie 'cmd': liefert "action(..) != "".
- Neues Attribut 'remove' innerhalb von <a> Elementen: Dieses kann (falls vorhanden) die Werte 'self' und 'all' annehmen. Es wird berücksichtigt, falls das Attribut 'href' leer ist (bzw. zu einem Leerstring evaluiert, s. Methode 'action'). In diesem Fall bedeutet der Wert 'self', dass das <a> Element entfernt wird, sein Inhalt aber beibehalten bleibt. Der Wert 'all' bedeutet, dass das <a> Element inklusive seinem Inhalt entfernt wird.
- Neues Attribut 'keep' innerhalb von  und <img> Elementen. Ist es vorhanden und sein Wert evaluiert nicht zu 'true', so wird das Element entfernt.

### Version 1.6

- Unterstützung von Element-Templates; diese Templates werden mit Hilfe des neuen Elementes `<bw:rule element="elementName"> ... </bw:rule>` definiert. Diese Templates werden dann auf die spezifizierten Elemente 'elementName' angewendet: Dabei wird ein passendes Element durch den Inhalt des <bw:rule>-


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Templates ersetzt. innerhalb des Templates kann dabei auf das zu ersetzende Element zugegriffen werden. Beispiel:

```
<!-- Template -->
<bw:rule element="kiddo">
<div id="{@id}"><input type="Text" value="{concat( 'Geburt-
stag: ', @value)}" /></div>
</bw:rule>

<!-- Zu transformieren -->
<kiddo id="mady" value="13.7.2001" >

<!-- Resultat der Transformation: -->
<div id="mady">
  <input type="Text" value="Geburtstag: 13.7.2001" />
</div>
```

### Version 1.5

- Ausgliederung des Java-DOM-Zugriffs in ein eigenes Paket biss-path
- Die Property Einträge biss.bml.DebugAccesses, biss.bml.DebugEvals und biss.bml.FieldCacheSize sind durch die analogen Einträge in biss_path.properties ersetzt worden: biss.path.DebugAccesses, biss.path.DebugEvals und biss.path.FieldCacheSize

### Version 1.4

- Voll qualifizierte Methodennamen; es ist jetzt möglich, innerhalb einer Expression einen temporären spezifischen Evaluator zur Methoden-Auswertung heranzuziehen. Dies geschieht mit der Syntax: 'java:className.methodName( ...)'; Beispiel:

```
java:biss.wise.bml.Evaluator.fullName( 'Partner')
```

- Fallunterscheidung (`<bw:switch>`): Mit dem Konstrukt

```
<bw:switch expr="x">
  <bw:case value="v1"> .. </bw:switch>
  <bw:case value="v2"> .. </bw:switch>
  ...
  <bw:case value="vn"> .. </bw:switch>
  <bw:default> .. </bw:default>
</bw:switch>
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

kann jetzt eine Fallunterscheidung realisiert werden.

- Erweitertes `<bw:include>` (neuer Modus `'replaceContent'`); mit dieser Operation ist es jetzt möglich nur den Inhalt eines adressierten Elements zu ersetzen. Beispiel:

```
<bw:include src="...">
  <bw:element mode="replaceContent"
path="html.body.table.tbody">
  <!-- Ersetzen des Inhaltes eines Elements -->
  <td> ... </td>
  <td> ... </td>
</bw:element>
```

- `<bw:var>` in `<bw:include>`; es ist jetzt möglich, Variablen auch unmittelbar innerhalb eines `<bw:include>` zu deklarieren.
- Erweiterte Aufklapp-Funktion-Generierung (mit Hilfe von `<bw:foreach>` mit `'maxn'` und `'autodiv'` Attributen); es wird jetzt auch in HTML-`<ul>`-Sektionen (bisher nur in HTML-`<table>`-Sektionen) eine entsprechende unsichtbare Sektion mit `id` gegeben durch den Wert des Attributs `'autodiv'` generiert.
- Fehler bei Timestamp-Attributen behoben; diese wurden ohne ihre Uhrzeitkomponente ausgewertet.
- Fehler bei Top-Level-Includes behoben; BML-Dokumente, deren Root-Element ein bw:include war, wurden fehlerhaft transformiert.
- Alternatives Element `<bw:page extends="url">`; dies ist eine alternative Schreibweise für `<bw:include src="url">`.
- Neue (HTML-spezifische) eingebaute Transformation; ein Ausdruck der Form `<bw:var class="cls" expr="x">...</bw:var>` wird implizit zu `<span class="cls"><bw:var expr="x">...</bw:var></span>`

**Version 1.3**

- Inheritance (erweitertes bw:include); es ist jetzt möglich, Teile (Elemente und Attribute) eines inkludierten XML-Fragments zu erweitern, ersetzen und zu löschen; dies geschieht mit den neuen Elementen bw:element und bw:attr; dabei wird der Zugriff auf das DOM der inkludierten XML-Datei ermöglicht.
- Ein Element (oder auch eine Liste von Elementen) kann dann über einen BML-Pfad-Ausdruck adressiert werden. Alternativ kann auch eine Adressierung über eine ID geschehen. Das neue BML-Elementes <bw:element> kann ein (oder mehrere) XML-Elemente adressieren und je nach Wert des Attributs 'mode' (Werte: "replace" "insert", "changeAttrs") die verschiedenen Modifikations-Operationen erlauben. Das neue BML-


© Copyright BISS GmbH, 2001-2012 All Rights reserved

Element <bw:attr> erlaubt es, Attribute zu ändern und zu löschen. Die verschiedenen <bw:element>-Anweisungen werden dabei sequentiell abgearbeitet: sie werden jeweils auf das Resultat der vorhergehenden Ersetzung 'angewendet'. Beispiel:

```
<bw:include src="...">
<bw:element mode="replace"
path="html.body.table.tbody.*td[pos()=0]"> <!-- Ersetzen ei-
nes Elements (ev. durch mehrere) -->
  <td> ... </td>
  <td> ... </td>
</bw:element>

<!-- Löschen von Elementen = Ersetzen durch nichts -->
<bw:element mode="replace" path="html.body.table.tbody.*td"
/>

<!-- Hinzufügen von Elementen -->
<!-- Opt. Angabe der Einfügeposition bei Listen durch
pos=".." -->
<bw:element mode="insert" path="html.body.table.tbody"
pos="0">
  <td> ... </td>
  <td> ... </td>
  <td> ... </td>
</bw:element>

<!-- Ändern von Attributen -->
<bw:element mode="changeAttrs"
path="html.body.table.tbody.td">
  <!-- Ersetzen /Hinzufügen eines Attributs -->
  <bw:attr name="colspan" value="3" />
  <!-- Löschen eines Attributs, entspricht value="" -->
  <bw:attr name="align" />
</bw:element>
  ...
</bw:include>
```

- Zugriff auf das Wurzel-Element mit der Methode `root()`.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

### Version 1.2.11

- Effizienterer Zugriff auf DataMember; hierzu wird ein einstellbarer Cache eingeführt. Seine Größe ist in biss_bml.properties mit Hilfe der Propertiy biss.bml.FieldCacheSize (Default = 100) einstellbar.

### Version 1.2.10

- Neue Evaluator-Methoden:

  `iff( cond, then, else)`
  liefert 'then', falls 'cond' == true, 'else' sonst.

  `beginTag( tag)`
  liefert '<tag>'

  `endTag( tag)`
  liefert '</tag>'

  `encloseIf( cond, left, middle, right)`
  liefert ('left' + 'middle' + 'right'), falls 'cond' == true, 'middle' sonst.

### Version 1.2.9

- Zugriff auf das aktuelle Element in einer for-each-Iteration mit der Methode current(). Beispiel:

```
<bw:var name="cur" expr="current()" />
    ... <bw:var expr="join( " ", $cur.Strasse, $cur.Hausnr)"
/>
```

### Version 1.2.8

Neue (per Javascript zugängliche) Evaluator-Methoden:

```
Object eval( Object context, String expr, boolean asString);
// Wertet 'expr' im Kontext 'context' aus und liefert das Er-
gebnis
// als String, falls 'asString' = true

String evalAsString( String expr);
// Entspricht eval( null, expr, true);
// Beispiel: evalAsString( "Kun-
de.AdresseListe.*Adresse[pos()=0].PLZ")
// liefert die PLZ der ersten Adresse des Kunden

String getItemAsString( String path);
```


© Copyright BISS GmbH, 2001-2012 All Rights reserved

```
// Entspricht getItem( null, path, true);
// Beispiel: getItemAsString( "Kunde.Vorname")
```

### Version 1.2.7

- Fehler beim Methoden-Aufruf mit null-Feldern behoben

### Version 1.2.6

- Fehler bei der formatierten XML-Ausgabe von Text gefolgt von Elements (<b>) behoben

### Version 1.2.5

- Neue String-Methoden:
  Methode `length( s)` liefert die Länge von s
  Methode `toLowerCase( s)` liefert s kleinbuchstabig
  Methode `toUpperCase( s)` liefert s großbuchstabig

### Version 1.2.4

- Neue mathematische Methoden
  Methode `bit( x, index)` liefert das index(0..31)-te Bit aus x
  Methode `ceil( x)` liefert die nächstgrößere ganze Zahl
  Methode `floor( x)` liefert die nächstkleinere ganze Zahl
  Methode `round( x)` liefert die nächste ganze Zahl

### Version 1.2.3

- Verbesserte Access-Warnungen in Listen-Ausdrücken

### Version 1.2.2

- Fehler bei der Behandlung von `style`-Attributen in ForEach-Schleifen behoben

### Version 1.2.1

- Fehler bei der Auswertung verschachtelter ForEach-Schleifen behoben
- Voreinstellung der Anzeige von Zugriffswarnungen und Auswertungen über Properties; mit Hilfe der Properties (im Propertyfile biss_bml.properties) biss.bml.DebugEvals (Default=false) und biss.bml.DebugAccesses (Default=false) kann die Ausschreibung von Zugriffswarnungen und Expression-Auswertungen eingeschaltet werden.

© Copyright BISS GmbH, 2001-2012 All Rights reserved

- Verbesserte Zugriffswarnungen

### Version 1.2

- Unterstützung von Includes; mit Hilfe des neuen BML-Elements `<bw:include src="filename" />` kann man an beliebiger Stelle beliebige Code-Segmente inkludieren. Die inkludierte Datei sollte selbst wieder eine wohlgeformte BML-Datei sein. Zu diesem Zweck wurde zusätzlich das neue BML-Element `<bw:group>` eingeführt, welches als Klammer um eine Liste von Elementen dienen kann (eine wohlgeformte BML-Datei kann ja nur ein Root-Element enthalten).

### Version 1.1.8

- Unterstützung von Lookup-Tables bei der Verwendung von non-XML-DOMS

### Version 1.1.7

- Teil-Auswertung von Attributwerten; XML-Attribute können nun an beliebigen Stellen Ausdrücke (eingeschlossen von { und }) enthalten, welche zur Transformationszeit ausgewertet werden. Beispiel:

```
<a href="javascript:doit( {.Name}, {pos() + 1})" />
<!-- Dies musste bisher umständlich konkateniert werden: →
<a href="{concat( 'javascript:doit(', .Name, ', ', pos()+1,
')')}" />
```

### Version 1.1.6

- Zusätzliche Methoden zur bedingten String-Konkatenation (liefern "", falls ein Argument = "" ist, andernfalls die Konkatenation ihrer Argumente):

```
concatNe( String s1, String s2)
concatNe( String s1, String s2, String s3)
concatNe( String s1, String s2, String s3, String s4)
concatNe( String s1, String s2, String s3, String s4, String
s5)
```

### Version 1.1.5

- Verbesserter DataAccessor: Berücksichtigt jetzt auch hasXX/isXX Methoden und erlaubt AbstractList zusätzlich zu Arrays als Listenrepräsentation
- Zusätzliche Methoden zur String-Quotierung:

```
doubleQuote()
```

liefert ein " Zeichen


© Copyright BISS GmbH, 2001-2012 All Rights reserved

`doubleQuote( String x)`
liefert x eingeschlossen in " ("x")
`singleQuote()`
liefert ein ' Zeichen
`singleQuote( String x)`
liefert x eingeschlossen in ' ('x')

- Verbessertes Tool-Verhalten im Fehlerfall

### Version 1.1.4

- Voreinstellung der angezeigten Dialoggröße über Properties; mit Hilfe der Properties (im Propertyfile biss_bml_tools.properties) biss.bml.tools.FullSize und biss.bml.tools.HalfSize kann die Größe des Design-Dialogs voreingestellt werden (wobei zwischen Modus 'Full size' und 'Half size' = nicht 'Full Size' unterschieden wird. Die Einträge erfolgen in Dialog Units (x, y, Breite, Höhe). Beispiel:

```
biss.bml.tools.FullSize=1000, 1000, 4700, 3000
biss.bml.tools.HalfSize=1000, 1000, 3000, 2000
```

- Anpassung an aktuelles BISS-AWT; verbesserter Zugriff auf non-XML Daten

### Version 1.1.3

- Verbesserter Zugriff auf non-XML Daten

### Version 1.1.2

- Umstieg auf ICE v5_1_1
- `autodiv` funktioniert wieder mit dem ICE DOM

### Version 1.1.1

- Einbindung der aktuellen ICE-Integration (ICE v5_1_0) inkl. BubbleHelp

### Version 1.1

- Austauschbares DOM; alternativ kann derzeit das JDOM und das ICE-DOM (Default) verwendet werden. Der Austausch geschieht durch Setzen der Umgebungsvariable im Property-File biss_bml.properties (für JDOM: `biss.bml.jdom.XMLAccessor`):

```
biss.bml.XMLAcc (Default=biss.bml.ice.XMLAccessor)
```

© Copyright BISS GmbH, 2001-2012 All Rights reserved

**Version 1.0**

- Initiale Version.

## Bekannte Fehler

- `count( $var)` funktioniert nicht, wenn `'var'` als Listen-Ausdruck definiert ist.


© Copyright BISS GmbH, 2001-2012 All Rights reserved